Winter semester 2021/22

344.175 VL: Natural Language Processing

Text Preprocessing and *n*-Gram Language Modeling

Navid Rekab-saz
navid.rekabsaz@jku.at

JOHANNES KEPLER
UNIVERSITY LINZ

Institute of
Computational
Perception

Agenda

- Text preprocessing & tokenization
- *N*-gram language models

Agenda

- **Text preprocessing & tokenization**
- *N*-gram language models

Text preprocessing

- Preprocessing is the first and a crucial step of NLP task
  - The objective of preprocessing is to clean/harmonize the text, reduce language fluctuations if necessary, and prepare the tokens for being processed in the next steps
  - Preprocessing partially addresses the issue with sparsity, why?

We will learn:

- Text cleaning/harmonization/reduction of fluctuations
  - Text normalization
  - Segmentation
  - Stop words
  - Stemming & Lemmatization
- Tokenization
  - Rule-based tokenization
  - Subword tokenization

Text Normalization

- Normalization harmonizes the written forms of the words with same meanings

- Some examples:
  - deleting periods
    - ***U.S.A.*** → ***USA***
  - deleting hyphens
    - ***anti-discriminatory*** → ***antidiscriminatory***
  - Accents
    - French ***résumé*** → ***resume***
  - Umlauts
    - German: ***Tuebingen*** → ***Tübingen***

Text Normalization

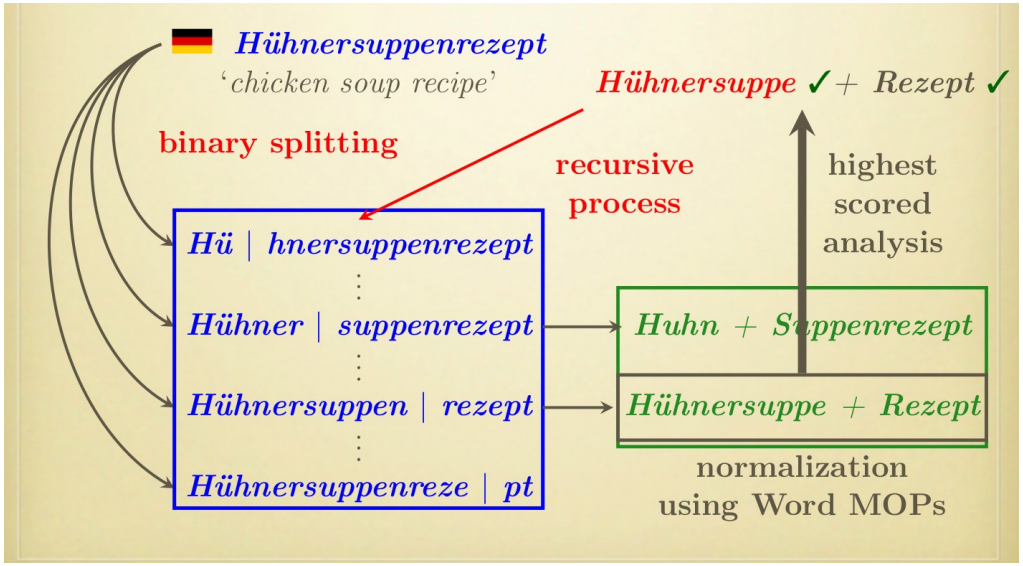
- Case folding: reduce all letters to lower case
  - It may cause ambiguity but typically helpful
    - **General Motors** vs. **general motors**
    - **Fed** vs. **fed**
    - **CAT** (City Airport Train) vs. **cat**

- Longstanding Google example:
  - Search **C.A.T.**

- Do the numbers, dates, etc. bring information?
  - If included, the dictionary size may explode!
  - Numbers and dates are commonly replaced by special tokens, e.g.
    - Numbers with <num>
    - Dates with <dates>

Segmentation

- Segmentation
  - Splitting a compound word into tokens
- French
  - ***L'ensemble*** → one token or two? ***L*** ? ***L'*** ? ***Le*** ?
- German compound nouns
  - ***Halsschlagader*** → ***Hals Schlag Ader***?
  - Compound words in German usually require **compound splitter**
  - A Possible algorithm (look in the link for more details):



https://www.ims.uni-stuttgart.de/forschung/ressourcen/werkzeuge/mcs/

Stop words

- Stop words
  - The commonest words, like ***the***, ***a***, ***and***, ***to***, ***be***
  - They carry little or no semantic information

- Stop words can also be important, especially in combination with other words, e.g.:
  - Phrases: ***"King of Denmark"***, ***"To be or not to be"***
  - Titles, etc.: ***"Let it be"***
  - Definitional purposes: "**flights to London**"

- Stop words are sometimes excluded from the corpus
  - Commonly in bag-of-words approaches

Stemming

- Morphemes in English consists of
  - Stem: the core meaning-bearing units
  - Affixes: pieces that adhere to stems

- A stemmer reduces words to their “stems” by crude affix chopping. Examples:
  - ***automate, automates, automatic, automation → automat***
  - ***for example compressed and compression are both accepted as equivalent to compress →***

    for exampl compress and compress ar both accept as

    equival to compress

Porter's stemmer

- Commonest algorithm for stemming English
  - consists of a set of grammatical commands

- Typical rules:
  - *sses* $\rightarrow$ *ss*
  - *ies* $\rightarrow$ *i*
  - *ational* $\rightarrow$ *ate*
  - *tional* $\rightarrow$ *tion*

Give it a try: https://text-processing.com/demo/stem/

Lemmatization

- A lemmatizer uses a knowledge resource (like WordNet) to find and replace base forms

- Lemmatzer reduces inflectional/variant forms to base forms. Examples:
  - ***am***, ***are***, ***is*** $\rightarrow$ ***be***
  - ***car***, ***cars***, ***car's***, ***cars'*** $\rightarrow$ ***car***
  - ***the boy's cars are different colors*** $\rightarrow$ ***the boy car be different color***

- Lemmatization versus Stemming:
  - Both reduce variation
  - Stemming is typically faster
  - Stemming may harm precision and increase ambiguity
  - If a given word does not exist in the knowledge resource, lemmatization may not be able to process it

Tokenization

- Tokenization
  - Splitting a running text into tokens

- Sample questions when tokenizing:
  - ***Finland's capital*** → ***Finland*** AND ***s***? ***Finlands***? ***Finland's***?
  - ***Hewlett-Packard*** → ***Hewlett*** and ***Packard*** as two tokens?
  - ***state-of-the-art*** → break up hyphenated sequence?
  - ***San Francisco*** → one token or two? ***San_Francisco*** or "***San***" and "***Francisco***"?
  - ***lowercase*** → ***lower-case***, ***lowercase***, or ***lower case***?
  - what's, don't → ***what is***, ***do not***?

Tokenization approaches

- Two general tokenization approaches
  - Rule-based tokenization
  - Subword tokenization

- Rule-based tokenization
  - Tokenization using a set of rules
  - needs language-specific knowledge
  - can become problematic in morphologically rich languages
  - common approach to tokenization
    - For instance, provided in libraries like spaCy and Moses, or by using Regular Expressions

Subword tokenization

- Motivating example:
  - "***structurally***" appears rarely, however, its meaning can be inferred from "***structure***" which may appear much more often in a corpus
  - Lemmatizers and stemmers turn "***structurally***" and "***structure***" to the same stem (like "***structur***"), they both
    1. require knowledge about the specific language in hand (like English or Swahili)
    2. remove the differences between these two words

- Subword tokenization uses corpus statistics to first create a vocabulary list of subwords, and then decomposes every word to these subwords.

- Subword tokenization does not need any knowledge about language but uses the occurrence statistics, extracted from a corpus.

Subword tokenization

Byte Pair Encoding (BPE)

- The core idea of BPE comes from information theory and compression

- BPE (or in general subword tokenizers) consist of two steps:
  1. **Training:** Learning a vocabulary list of subwords from a given corpus
  2. **Tokenization (decoding):** tokenize a given text using the stored subwords vocabulary list

Reference: Sennrich, R., Haddow, B., & Birch, A.. Neural Machine Translation of Rare Words with Subword Units. In *proceedings of ACL (2016)*

Byte Pair Encoding (BPE)

Sketch of training:

1. Pre-tokenize the training corpus, for instance simply by splitting on white spaces
2. Start from a vocabulary list with all single characters
3. Create a dictionary of words and counts from the pre-tokenized training corpus
4. Add special character "_" to the end of each word in the dictionary
5. Expand the vocabulary list:
   - Find the most frequent pair of characters in the dictionary of words
   - Merge the characters, and add them to the vocabulary list
   - Repeat step 5 till some limits on vocabulary size are reached

Learn more at "*Speech and Language Processing (3rd ed.) D. Jurafsky and J. H. Martin*", section 2.4.3.
https://web.stanford.edu/~jurafsky/slp3/2.pdf (the resource of the example)

Byte Pair Encoding – example

- Consider a tiny training corpus that leads to the following dictionary and vocabulary list

	dictionary	vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w
2	l o w e s t _	
6	n e w e r _	
3	w i d e r _	
2	n e w _	

	dictionary	vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w
2	l o w e s t _	
6	n e w e r _	
3	w i d e r _	
2	n e w _	

First merge

	dictionary	vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, r_
2	l o w e s t _	
6	n e w e r_	
3	w i d e r_	
2	n e w _	

Next merge

	dictionary	vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, r_, er_
2	l o w e s t _	
6	n e w er_	
3	w i d er_	
2	n e w _	

	dictionary	vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, r_, er_
2	l o w e s t _	
6	n e w er_	
3	w i d er_	
2	n e w _	

Next merge

	dictionary	vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew
2	l o w e s t _	
6	n ew er_	
3	w i d er_	
2	n ew _	

If we continue

Merge	Current Vocabulary
(n, ew)	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new
(l, o'	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new, lo
(lo, w)	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new, lo, low
(new, er_)	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new, lo, low, newer_
(low, _)	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new, lo, low, newer_, low_

WordPiece tokenization

- WordPiece is a descendent of BPE and has the following differences:

- Selecting character pairs for merging in WordPiece is based on "*minimizing the language model likelihood of the training data*"*

- WordPiece indicates internal subwords with "*##*" special symbol
  - E.g. "***unavoidable***" → ["***un***", "***##avoid***", "***##able***"]

* For more details look into the original paper:
Schuster, M., & Nakajima, K. (2012). Japanese and Korean voice search.
In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*

WordPiece tokenization

- Tokenization (decoding) is done using MaxMatch algorithm
  - A greedy longest-match-first algorithm
  - MaxMatch choses the longest token in the vocabulary that matches the given word
  - After a match, it repeats the previous step with the remainder of the word

```
function MAXMATCH(string, dictionary) returns list of tokens T

    if string is empty
        return empty list
    for i ← length(sentence) downto 1
        firstword = first i chars of sentence
        remainder = rest of sentence
        if InDictionary(firstword, dictionary)
            return list(firstword, MaxMatch(remainder,dictionary) )
```

WordPiece tokenization

- WordPiece with MaxMatch decoding is used in BERT

- Example *"**natural language processing**"* →

First pre-tokenization: [*"**natural**"*, *"**language**"*, *"**processing**"*] →

Then subword tokenization: [*"**natural**"*, *"**lang**"*, *"**##uage**"*, *"**process**"*, *"**##ing**"*]

```
function MAXMATCH(string, dictionary) returns list of tokens T

    if string is empty
        return empty list
    for i ← length(sentence) downto 1
        firstword = first i chars of sentence
        remainder = rest of sentence
        if InDictionary(firstword, dictionary)
            return list(firstword, MaxMatch(remainder,dictionary) )
```

NLP Libraries

AllenNLP

gensim

Stanford CoreNLP

spaCy

huggingface.co

polyglot

PYTORCH

Agenda

- Text preprocessing & tokenization
- ***N*-gram language models**

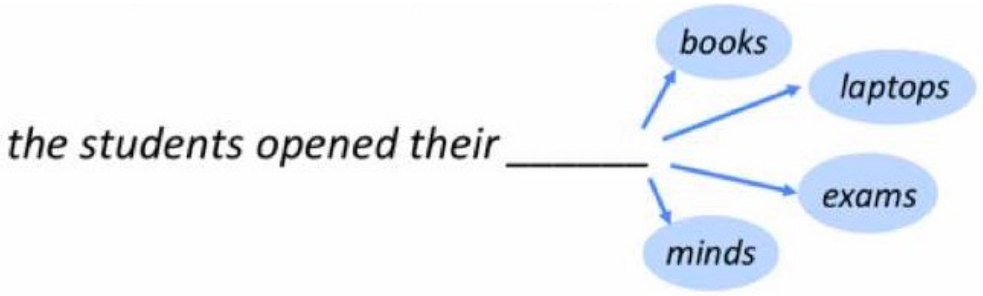
Some slides are adopted from http://web.stanford.edu/class/cs224n/ & https://web.stanford.edu/~jurafsky/slp3/3.pdf

Language Modeling

- Language Modeling is the task of predicting a word (or a subword or character) given a context:

$$P(v|\text{context})$$

- A Language Model (LM) can answer the questions like



$P(v|\textit{the students opened their})$

Language Modeling – formal definition

- Given a sequence of words $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$, a language model calculates the probability distribution of next word $x^{(t+1)}$ over all words in vocabulary

$$P(x^{(t+1)} | x^{(t)}, x^{(t-1)}, \ldots, x^{(1)})$$

x is any word in the vocabulary $\mathbb{V} = \{v1, v2, \ldots, vN\}$

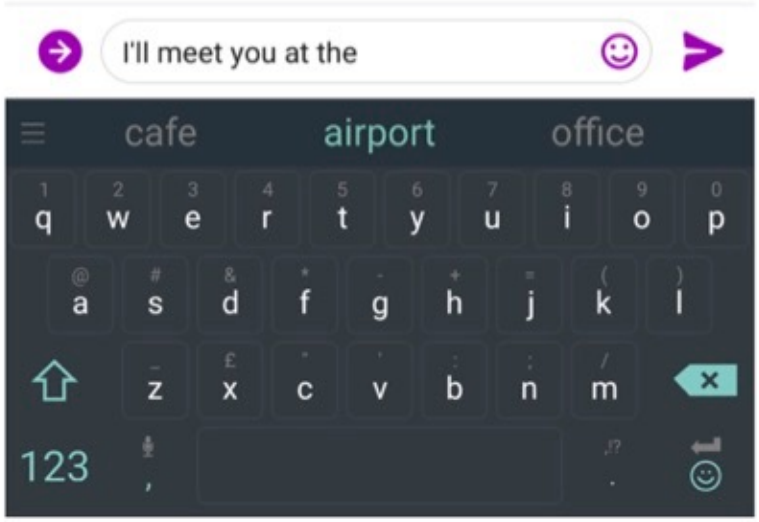
☞ Note: this is the definition of directed left-to-right language models.

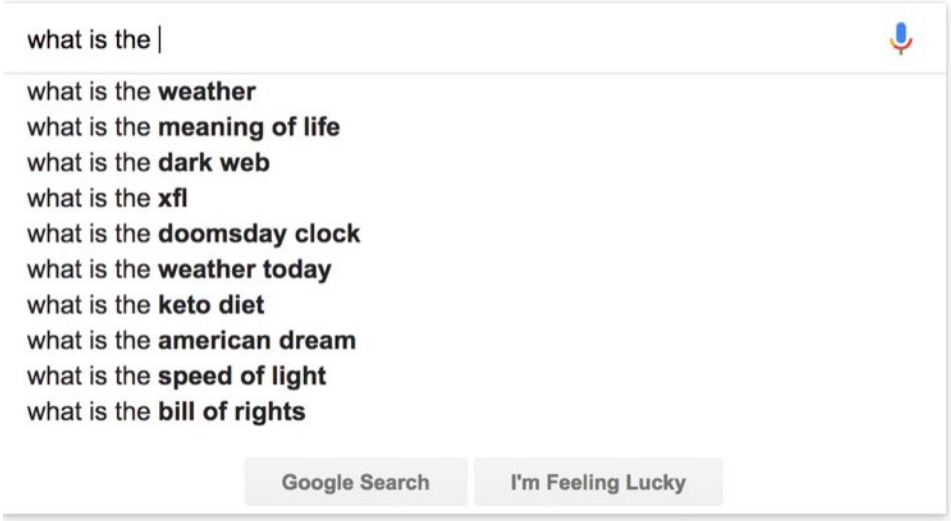
Why Language Modeling?

- Language Modeling is a benchmark task that helps us measure our progress on understanding language

- LMs are a subcomponent of many NLP tasks, especially those involving generating text or estimating the probability of text:
  - Predictive typing
  - Spelling/grammar correction
  - Automatic speech recognition (ASR)
  - Handwriting recognition
  - Machine translation
  - Summarization
  - Dialogue /chatbots
  - etc.

Direct usages for next word prediction





[link]

Probability of a Text

- A Language Model can also assign probability to the validity a piece of text
  - How probable it is that a sentence appears in a language.

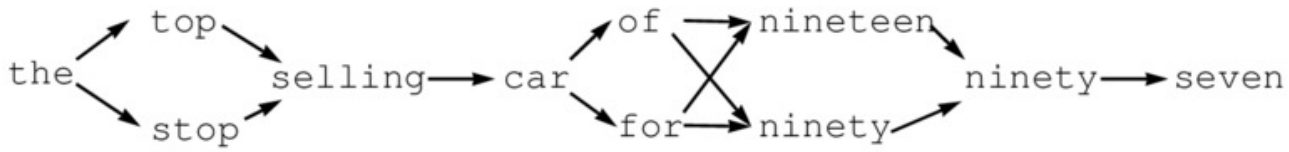
$$P\left(x^{(1)}, \ldots, x^{(T)}\right) = ?$$

- According to a (directed left-to-right) Language Model, the probability of a given text is computed by:

$$P\left(x^{(1)}, \ldots, x^{(T)}\right) = P\left(x^{(1)}\right) \times P\left(x^{(2)} \middle| x^{(1)}\right) \times \cdots \times P\left(x^{(T)} \middle| x^{(T-1)}, \ldots, x^{(1)}\right)$$

$$P\left(x^{(1)}, \ldots, x^{(T)}\right) = \prod_{t=1}^{T} P\left(x^{(t)} | x^{(t-1)}, \ldots, x^{(1)}\right)$$

Usage in Automatic Speech Recognition (ASR)



Input Lattice

Lattice Rescoring Tool

```
the top selling car of nineteen ninety seven
the top selling car of ninety ninety seven
the top selling car for ninety ninety seven
the stop selling car for nineteen ninety seven
```

N-Best List

n-gram Language Model

- Recall: a *n*-gram is a chunk of *n* consecutive words.

the students opened their _____

- unigrams: *"the", "students", "opened", "their"*
- bigrams: *"the students", "students opened", "opened their"*
- trigrams: *"the students opened", "students opened their"*
- 4-grams: *"the students opened their"*

- A *n*-gram Language Model collects frequency statistics of different *n*-grams in a corpus, and use these to calculate probabilities

N-gram LM as a conditional probability

- Markov assumption: decision at time t depends only on the current state

- In *n*-gram Language Model: predicting $x^{(t+1)}$ depends on preceding *n-1* words

- Without Markovian assumption:

$$P(x^{(t+1)}|x^{(t)}, x^{(t-1)}, \ldots, x^{(1)})$$

- *n*-gram Language Model:

$$P(x^{(t+1)}|\underbrace{x^{(t)}, x^{(t-1)}, \ldots, x^{(t-n+2)}}_{n\text{-}1 \text{ words}})$$

N-gram LM as a conditional probability

- Based on definition of conditional probability:

$$P\left(x^{(t+1)} \middle| x^{(t)}, \ldots, x^{(t-n+2)}\right) = \frac{P\left(x^{(t+1)}, x^{(t)}, \ldots, x^{(t-n+2)}\right)}{P\left(x^{(t)}, \ldots, x^{(t-n+2)}\right)}$$

- The *n*-gram probability is calculated by counting *n*-grams and [*n–1*]-grams in a large corpus of text:

$$P\left(x^{(t+1)} \middle| x^{(t)}, \ldots, x^{(t-n+2)}\right) \approx \frac{\text{count}\left(x^{(t+1)}, x^{(t)}, \ldots, x^{(t-n+2)}\right)}{\text{count}\left(x^{(t)}, \ldots, x^{(t-n+2)}\right)}$$

Example

- Example: learning a 4-gram Language Model

~~*as the exam clerk started the clock, the*~~ *students opened their* _____

condition on this

$$P(v|\textit{students opened their}) = \frac{P(\textit{students opened their } v)}{P(\textit{students opened their})}$$

- For example, suppose that in the corpus:
  - "*students opened their*" occurred 1000 times
  - "*students opened their books*" occurred 400 times
    - $P(\textit{books} \mid \textit{students opened their}) = 0.4$
  - "*students opened their exams*" occurred 100 times
    - $P(\textit{exams} \mid \textit{students opened their}) = 0.1$

Example – a bigram LM

- Trained on the data of a restaurant dialogue system

Bigram counts:

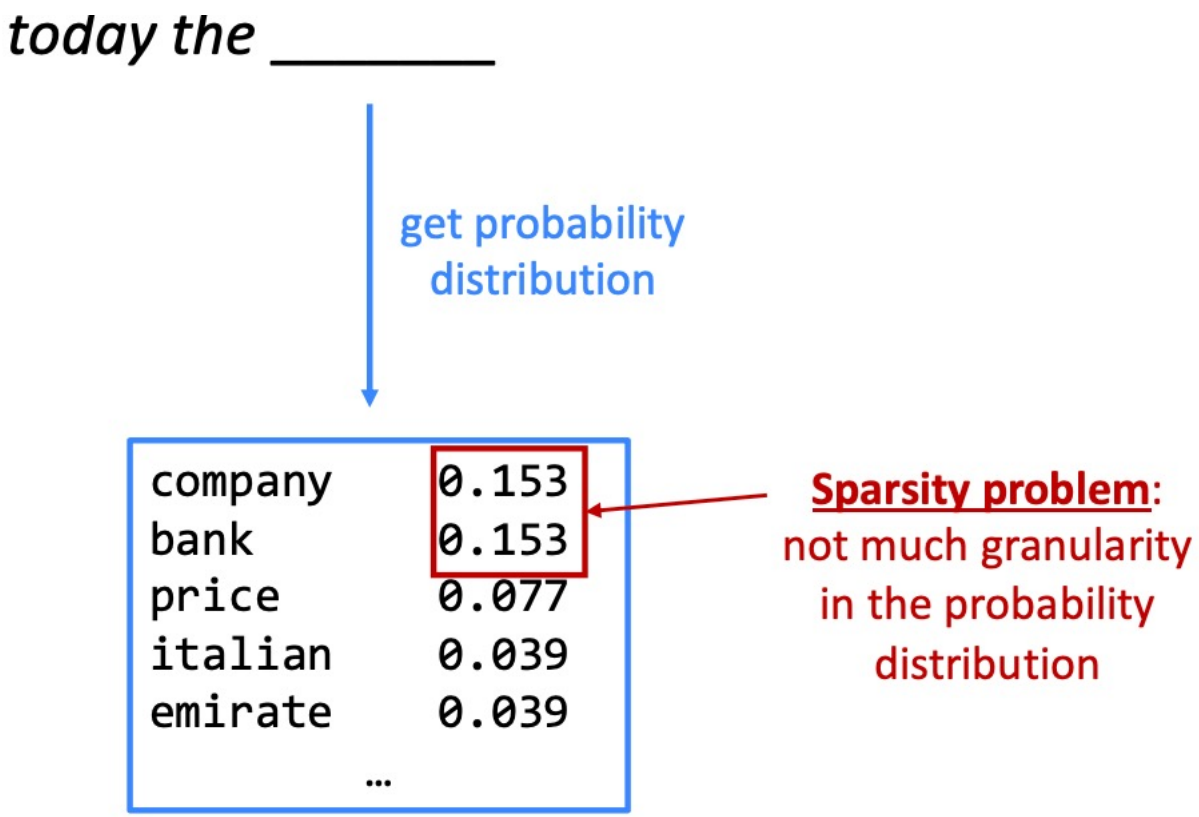
	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Bigram LM:

	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

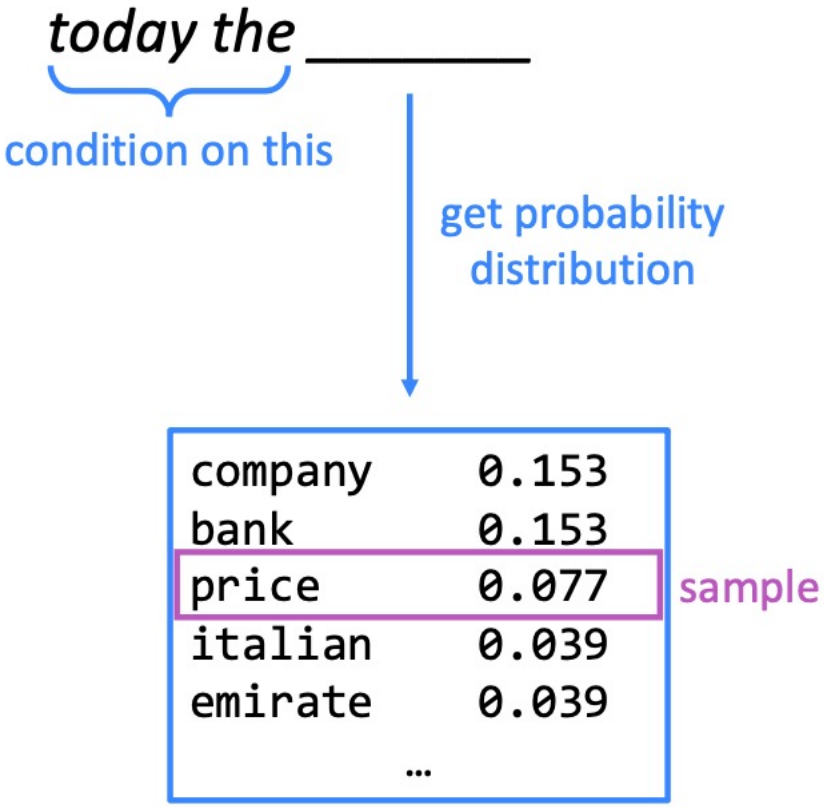
Generating text

- A trigram LM trained on Reuters corpus (1.7 M words)



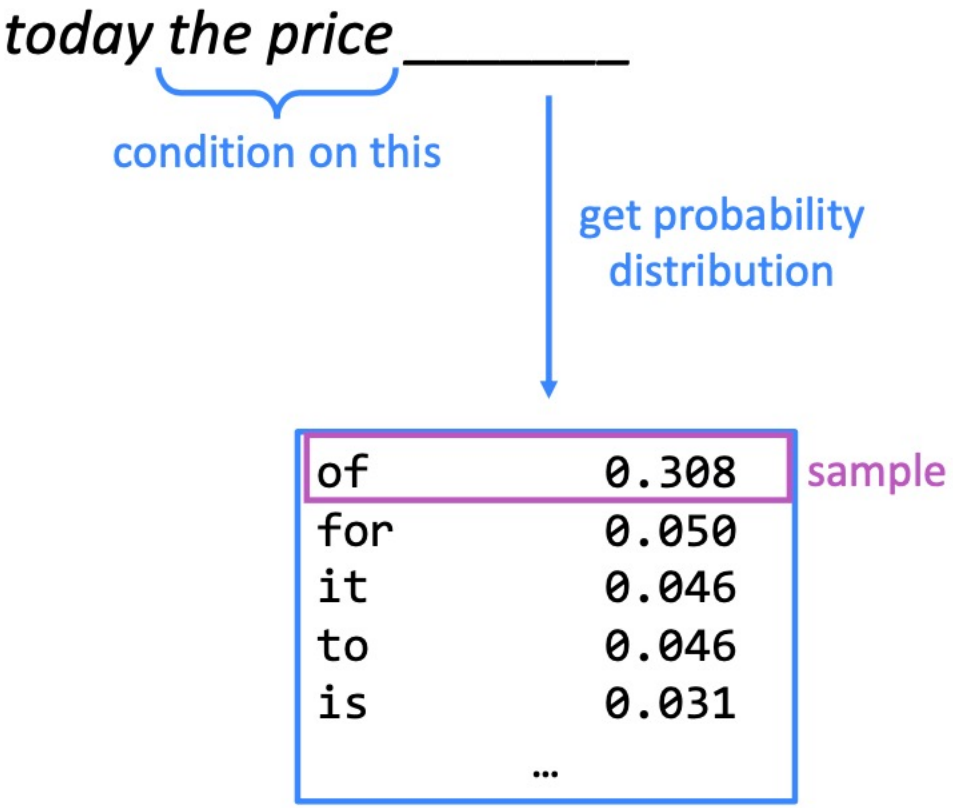
Generating text

- Generating text by sampling from the probability distributions



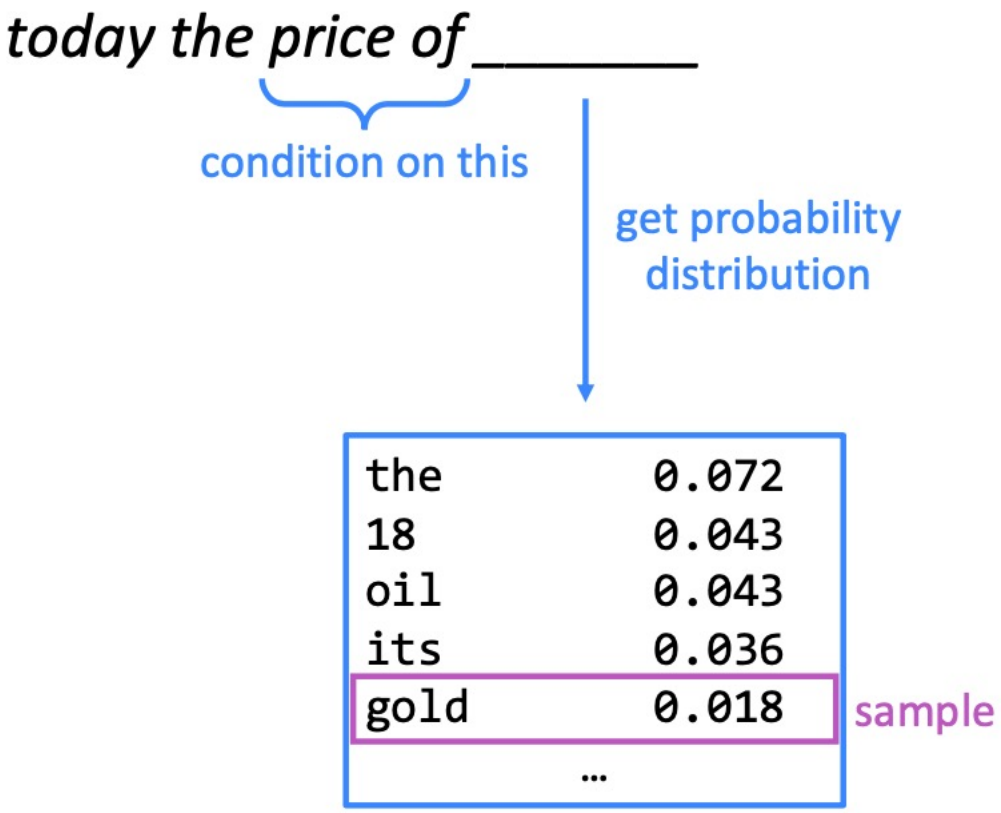
Generating text

- Generating text by sampling from the probability distributions



Generating text

- Generating text by sampling from the probability distributions



Generating text

- Generating text by sampling from the probability distributions

today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks , sept 30 end primary 76 cts a share .

- Decently good in syntax … but incoherent!
- Increasing *n* makes the text more coherent but also intensifies the discussed issues

Generating text

- *N*-gram LMs trained on Shakespeare's works

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.

Generating text

- *N*-gram LMs trained on Wall Street Journal

1 gram	Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives
2 gram	Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her
3 gram	They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

N-gram LMs – limitations

- **Sparsity**
  - What if the denominator never occurred in corpus?
    - Backoff: Probability is calculated for lower *n*-grams.
      - E.g., in „*students opened their* v“, if „*students opened their*“ does not exist, language model probability is calculated for “*opened their*”
      - Trigram is backed off to let a bigram do the job!
  - What if the nominator never occurred in corpus?
    - Approached by various smoothing methods

Laplace smoothing

- Add a small number like $\delta = 1$ to the count of all words:

$$\boldsymbol{P}\left(\boldsymbol{x}^{(t+1)} \middle| \boldsymbol{x}^{(t)}, \dots, \boldsymbol{x}^{(t-n+2)}\right) = \frac{\#\left(\boldsymbol{x}^{(t+1)}, \boldsymbol{x}^{(t)}, \dots, \boldsymbol{x}^{(t-n+2)}\right)}{\#\left(\boldsymbol{x}^{(t)}, \dots, \boldsymbol{x}^{(t-n+2)}\right)}$$

$$P\left(x^{(t+1)} \middle| x^{(t)}, \dots, x^{(t-n+2)}\right) = \frac{\#\left(x^{(t+1)}, x^{(t)}, \dots, x^{(t-n+2)}\right)}{\sum_{v \subset V} \#\left(x^{(t)}, \dots, x^{(t-n+2)}, v\right)}$$

$$P_{Laplace}\left(x^{(t+1)} \middle| x^{(t)}, \dots, x^{(t-n+2)}\right) = \frac{\#\left(x^{(t+1)}, x^{(t)}, \dots, x^{(t-n+2)}\right) + 1}{\sum_{v \subset V}\left(\#\left(x^{(t)}, \dots, x^{(t-n+2)}, v\right) + 1\right)}$$

$$P_{Laplace}\left(x^{(t+1)} \middle| x^{(t)}, \dots, x^{(t-n+2)}\right) = \frac{\#\left(x^{(t+1)}, x^{(t)}, \dots, x^{(t-n+2)}\right) + 1}{\sum_{v \subset V}\left(\#\left(x^{(t)}, \dots, x^{(t-n+2)}, v\right)\right) + |V|}$$

$$\boldsymbol{P_{Laplace}}\left(\boldsymbol{x}^{(t+1)} \middle| \boldsymbol{x}^{(t)}, \dots, \boldsymbol{x}^{(t-n+2)}\right) = \frac{\#\left(\boldsymbol{x}^{(t+1)}, \boldsymbol{x}^{(t)}, \dots, \boldsymbol{x}^{(t-n+2)}\right) + \boldsymbol{1}}{\#\left(\boldsymbol{x}^{(t)}, \dots, \boldsymbol{x}^{(t-n+2)}\right) + |\boldsymbol{V}|}$$

Laplace smoothing example

Original 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

Bigram counts added by 1:

	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 smoothing example

LM without smoothing:

	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

LM with Laplace smoothing:

	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

N-gram LMs – limitations

- **Sparsity**
  - What if the denominator never occurred in corpus?
    - Backoff: Probability is calculated for lower *n*-grams.
      - E.g., in „*students opened their* v“, if „*students opened their*“ does not exist, language model probability is calculated for “*opened their*”
      - Trigram is backed off to let a bigram do the job!
  - What if the nominator never occurred in corpus?
    - Approached by various smoothing methods
  - Sparsity issue becomes even more prominent in higher *n*-gram!

Learn more: https://web.stanford.edu/~jurafsky/slp3/3.pdf

N-gram LMs – limitations

- **Storage**
  - An *n*-gram language model needs to store all levels of *n*-grams, from uni- to *n*-gram, observed in the corpus
  - Increasing *n* radically worsens the storage problem!

- **No understanding of tokens relations**
  - Semantic and syntactic relations between words are fully ignored
    - *"book"* and "*books*" or *"car"* and "*automobile*" are treated completely separately