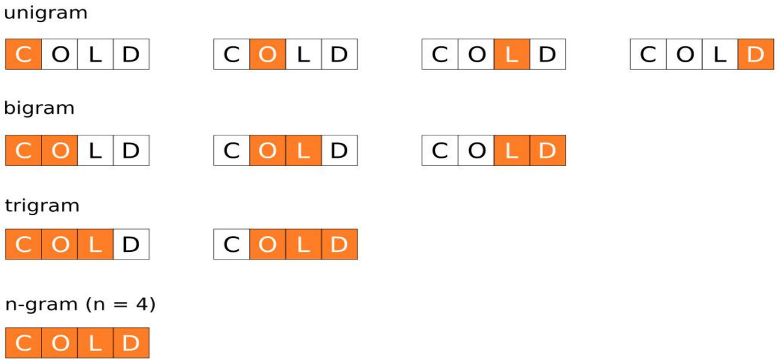
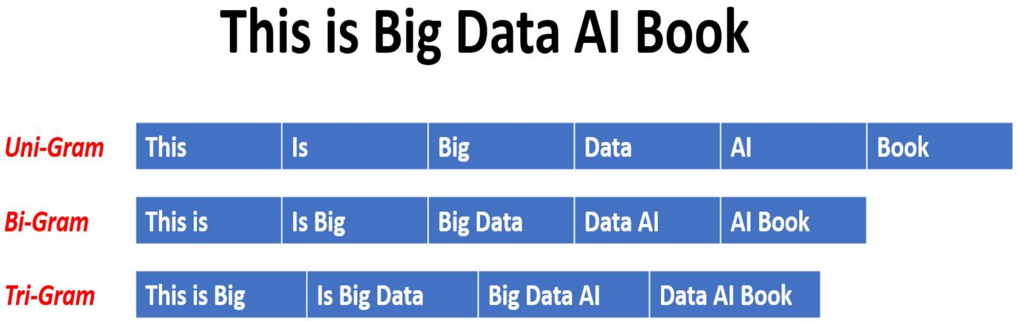
# Statistical Language Modeling with N-grams in Python

By Olha Diakonova

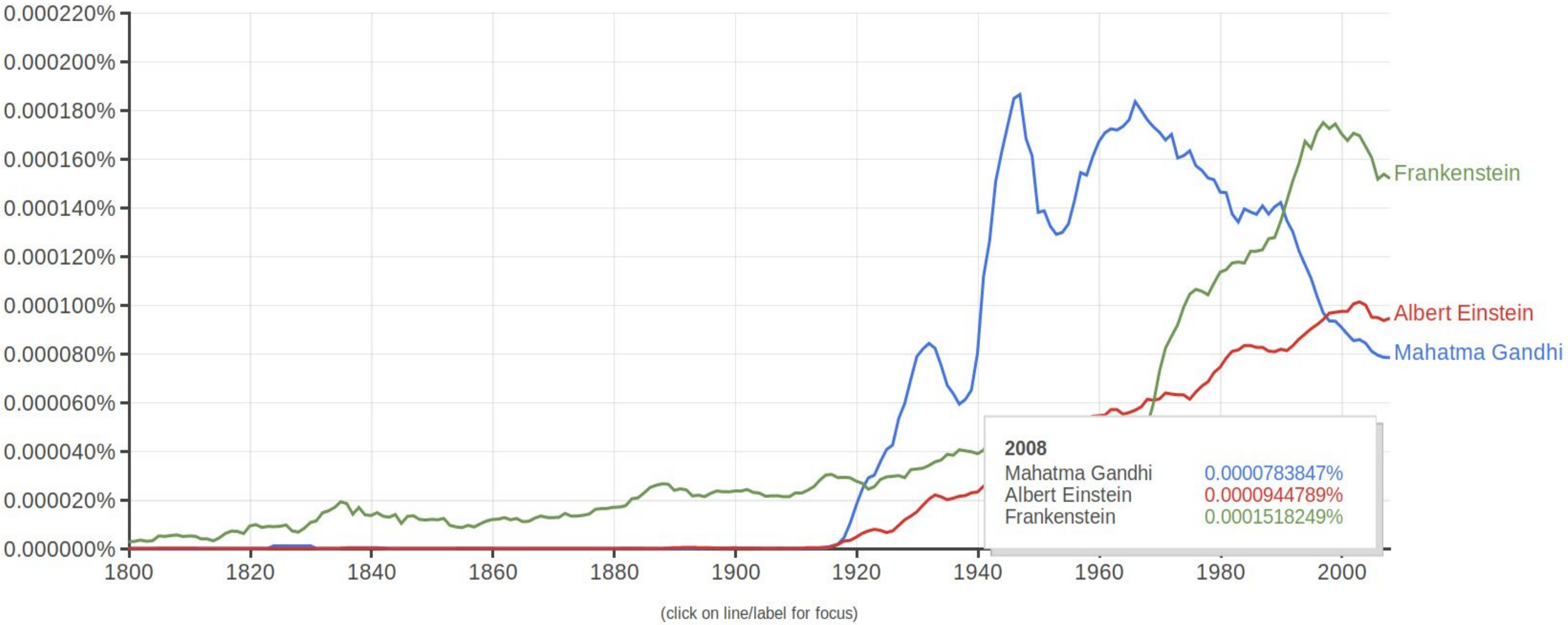
# What are n-grams




- Sequences of *n* language units
- Probabilistic language models based on such sequences
- Collected from a text or speech corpus
- Units can be characters, sounds, syllables, words
- Probability of $n^{th}$ element based on preceding elements
- Probability of the whole sequence

# Google N-gram Viewer



# Probabilities for language modeling

- Two related tasks:
- Probability of a word *w* given history *h*

  $P(w|h) = P(w, h) / P(h)$

$P(\textit{that}|\textit{water is so transparent}) = C(\textit{water is so transparent that}) / C(\textit{water is so transparent})$

- Probability of the whole sentence
- Chain rule of probability

  $P(w^n_1) = P(w_1)\ P(w_2)|P(w_1)\ P(w_3)|P(w^2_1)\ \ldots\ P(w_n|w^{n-1}_1) =$

  $= \prod_{k=1} P(w_k|w^{k-1}_1)$

- Not very helpful: no way to compute the exact probability of all preceding words

# Probabilities for language modeling

- **Markov assumption**: the intuition behind n-grams
- Probability of an element only depends on one or a couple of previous elements
- Approximate the history by just the last few words

$$P(w_n|w^{n-1}_1) \approx P(w_n|w^{n-1}_{n-N+1})$$

- N-grams are an insufficient language model:

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

- But we can still get away with it in a lot of cases

# What are n-grams used for

- Spell checking

  *The office is about 15 **minuets** away.*
  *P(about 15 **minutes** away) > P(about 15 **minuets** away)*

- Text autocomplete
- Speech recognition

  *P(I saw a van) > P(eyes awe of an)*

- Handwriting recognition
- Automatic language detection

| | □C | □L | □Z | Th |
|---|---|---|---|---|
| **English** | 0.75 | 0.47 | 0.02 | 0.74 |
| **German** | 0.10 | 0.37 | 0.53 | 0.03 |
| **French** | 0.38 | 0.69 | 0.01 | 0.01 |

- Machine translation

  *P(**high** winds tonight) > P(**large** winds tonight)*

- Text generation
- Text similarity detection

# Implementing n-grams

- Unigrams: sequences of 1 element
- Elements are independent
- Concept is similar to bag-of-words
- Can be used for a dataset with sparse features or as a fallback option

```
sentence = 'This is an awesome sentence .'
char_unigrams = [ch for ch in sentence]
word_unigrams = [w for w in sentence.split()]


print(char_unigrams)
print(word_unigrams)
```

```
['T', 'h', 'i', 's', ' ', 'i', 's', ' ', 'a', 'n', ' ',
'a', 'w', 'e', 's', 'o', 'm', 'e', ' ', 's', 'e', 'n', 't',
'e', 'n', 'c', 'e', '.']
['This', 'is', 'an', 'awesome', 'sentence.']
```

# Implementing n-grams

- Bigrams: sequences of 2 elements
- Trigrams: sequences of 3 elements

```
from nltk import bigrams

sentence = 'This is an awesome sentence .'

print(list(bigrams(sentence.split())))
print(list(trigrams(sentence.split())))
```

```
Bigrams: [('This', 'is'), ('is', 'an'), ('an',
'awesome'), ('awesome', 'sentence'),
('sentence', '.')]
Trigrams: [('This', 'is', 'an'), ('is', 'an',
'awesome'), ('an', 'awesome', 'sentence'),
('awesome', 'sentence', '.')]
```

# Implementing n-grams

- Generalized way of making n-grams for any n
- 4- and 5-grams: produce a more connected text, but there is a danger of overfitting

```
sent = "This is an awesome sentence for making n-grams ."

def make ngrams(text, n):
   tokens = text.split()
   ngrams = [tuple(tokens[i:i+n]) for i in
range(len(tokens)-n+1)]
   return ngrams

for ngram in make_ngrams(sent, 5):
   print(ngram)

('This', 'is', 'an', 'awesome', 'sentence')
('is', 'an', 'awesome', 'sentence', 'for')
('an', 'awesome', 'sentence', 'for', 'making')
('awesome', 'sentence', 'for', 'making', 'n-grams')
('sentence', 'for', 'making', 'n-grams', '.')
```

# Implementing n-grams

- NLTK implementation
- Paddings at string start & end
- Ensure each element of the sequence occurs at all positions
- Keep the probability distribution correct

```
from nltk import ngrams

sent = "This is an awesome sentence ."
grams = ngrams(sent.split(),5, pad right=True,
                              right pad symbol='</s>',
                              pad left=True,
                              left_pad_symbol='<s>',)

for g in grams:
   print(g)

('<s>', '<s>', '<s>', '<s>', 'This')
('<s>', '<s>', '<s>', 'This', 'is')
('<s>', '<s>', 'This', 'is', 'an')
('<s>', 'This', 'is', 'an', 'awesome')
('This', 'is', 'an', 'awesome', 'sentence')
('is', 'an', 'awesome', 'sentence', '.')
('an', 'awesome', 'sentence', '.', '</s>')
('awesome', 'sentence', '.', '</s>', '</s>')
('sentence', '.', '</s>', '</s>', '</s>')
('.', '</s>', '</s>', '</s>', '</s>')
```

# Dealing with zeros

- What if we see things that never occur in the corpus?
- That’s where **smoothing** comes in
- Steal probability mass from the present n-grams and share it with the ones that never occur
- OOV - out of vocabulary words
- Add-one estimation aka Laplace smoothing
- Backoff  and interpolation
- Good-Turing smoothing
- Kneser-Ney smoothing

# Practice time

- Let’s generate text using an n-gram model!
- The Witcher aka Geralt of Rivia quotes

# References


1. Dan Jurafsky. N-gram Language Models - Chapter from Speech and Language Processing: https://web.stanford.edu/~jurafsky/slp3/3.pdf
2. Dan Jurafsky lectures: https://youtu.be/hB2ShMlwTyc
3. GitHub: https://github.com/olga-black/ngrams-pykonik
4. Bartosz Ziołko, Dawid Skurzok. N-grams Model For Polish: http://www.dsp.agh.edu.pl/_media/pl:resources:ngram-docu.pdf
5. Corpus source: https://witcher.fandom.com/wiki/Geralt_of_Rivia/Quotes
6. Corpus source: https://www.magicalquote.com/character/geralt-of-rivia/

# About me


- Olha Diakonova
- Advertisement Analyst for Cognizant @ Google
- olha.v.diakonova@gmail.com
- GitHub: https://github.com/olga-black
- Pykonik Slack: Olha

# Thank you very much!