CMPU 366 · Natural Language Processing

Model Evaluation and Smoothing

17 September 2025

Where are we?

The idea of a statistical *language model* (LM) is to compute the probability of a sequence of words,

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

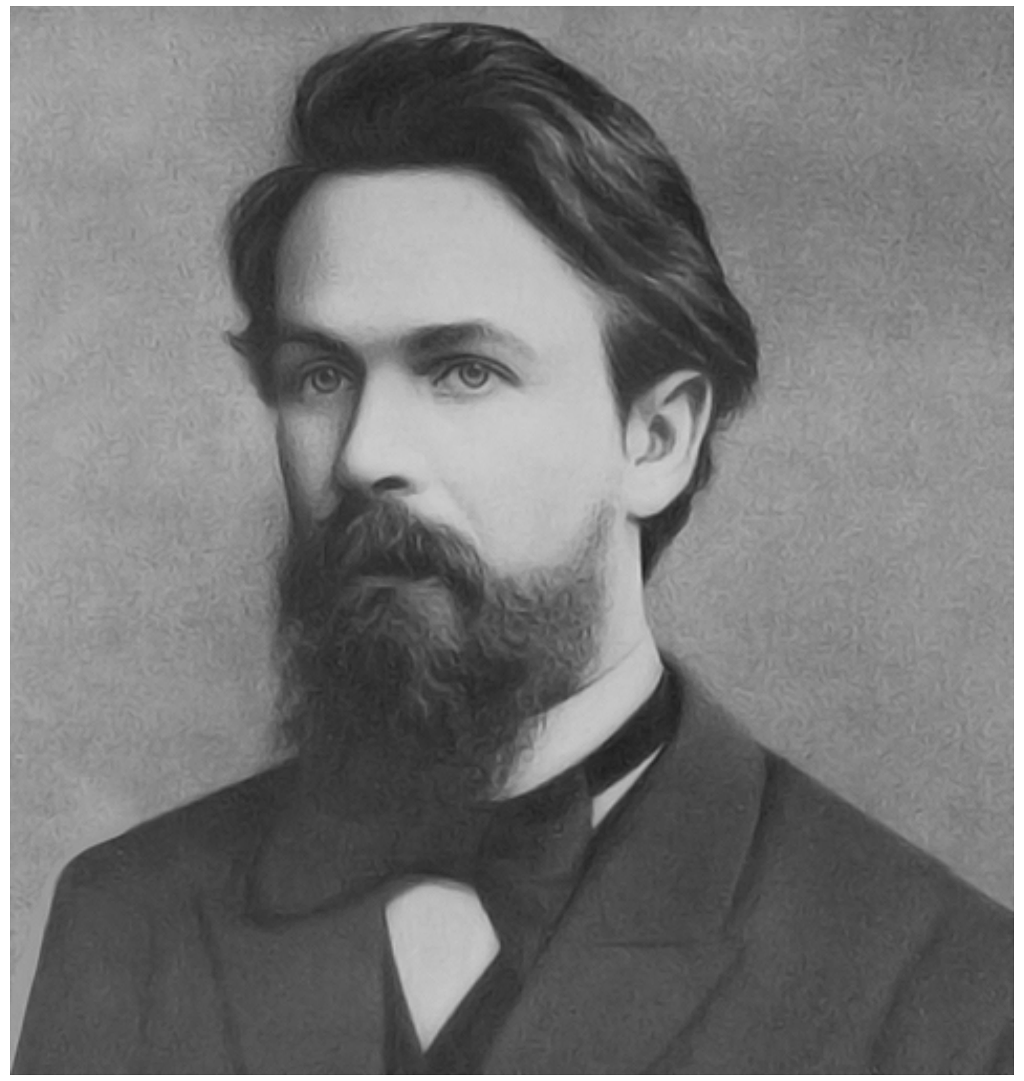
or the probability of an upcoming word given previous words,

$$P(w_\text{n} \mid w_1, w_2, w_3, \ldots, w_{n-1}).$$

In practice, we make a simplifying ***Markov assumption*** that we can predict the probability of a future event without looking too far into the past, e.g.,

$$P(\textit{blue} \mid \textit{The}, \textit{water}, \textit{of}, \textit{Walden}, \textit{Pond}, \textit{is}, \textit{so}, \textit{beautifully})$$

$$\approx P(\textit{blue} \mid \textit{so}, \textit{beautifully})$$



Andrei Markov

We estimate the true probability of a sequence of tokens using ***n-grams*** – sequences of tokens that are always *n* tokens long, e.g., bigrams (n=2) or trigrams (n=3).

Generating text

Generate random sentences:

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 randomly choose </s>.

<s> I

Generate random sentences:

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 randomly choose </s>.

<s> *I want*

Generate random sentences:

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 randomly choose </s>.

<s> I want to

Generate random sentences:

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 randomly choose </s>.

<s> I want to eat

Generate random sentences:

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 randomly choose </s>.

<s> I want to ***eat food***

Generate random sentences:

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 randomly choose </s>.

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

Approximating Shakespeare

Approximating Shakespeare

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

Approximating Shakespeare

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.*

Approximating Shakespeare

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.*

Approximating Shakespeare

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.*	*It cannot be but so.*

Fitting and overfitting

We estimate the probabilities of *n*-grams using the ***maximum likelihood estimate*** (MLE) – the estimate that maximizes the likelihood of the training data given the model.

For unigram probabilities,

that's the fraction of times the word occurs in the corpus:

$$P(w_i) = \frac{C(w_i)}{|V|}$$

For bigram probabilities,

that's the number of times that word follows the other word divided by the number of times the other word occurs in the corpus:

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

Probability is assigned *exactly* based on the *n*-gram count in the training corpus

Anything not found in the training corpus gets probability 0.

Shakespeare as a corpus

884 647 tokens

29 066 types

Shakespeare produced 300 000 bigram types out of 844 million possible bigrams.

Using MLE, 99.96% of the possible bigrams will have a probability of 0.

The perils of overfitting

N-grams only work well for word prediction if the test corpus looks like the training corpus.

- In real life, it often doesn't.
- We need to train robust models that generalize!

If we assume every sequence of words we'll ever see occurs in our training data, that's a kind of *overfitting*.

We want to learn from the training data but also generalize.

Smoothing and generalization

Laplace smoothing

Problem: Sparsity

Training set:

… *denied the allegations* (3×)

… *denied the reports* (2×)

… *denied the claims* (1×)

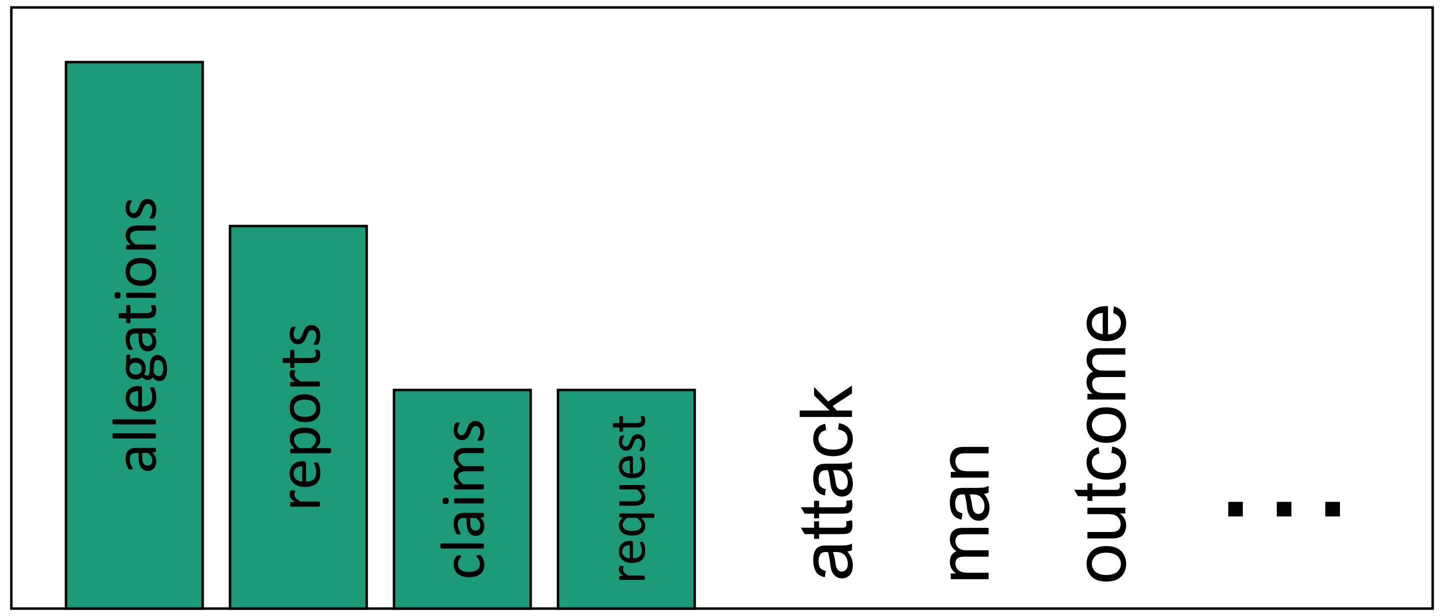
… *denied the request* (1×)

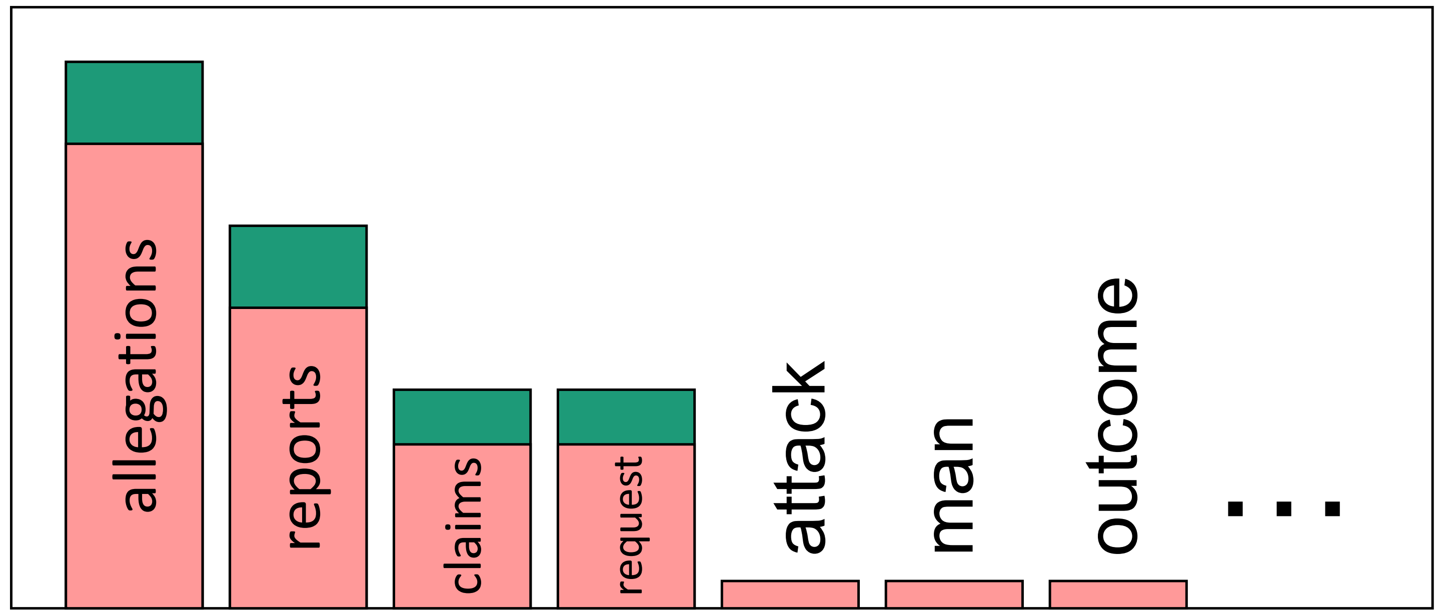
Test set:

… *denied the attack*

… *denied the man*

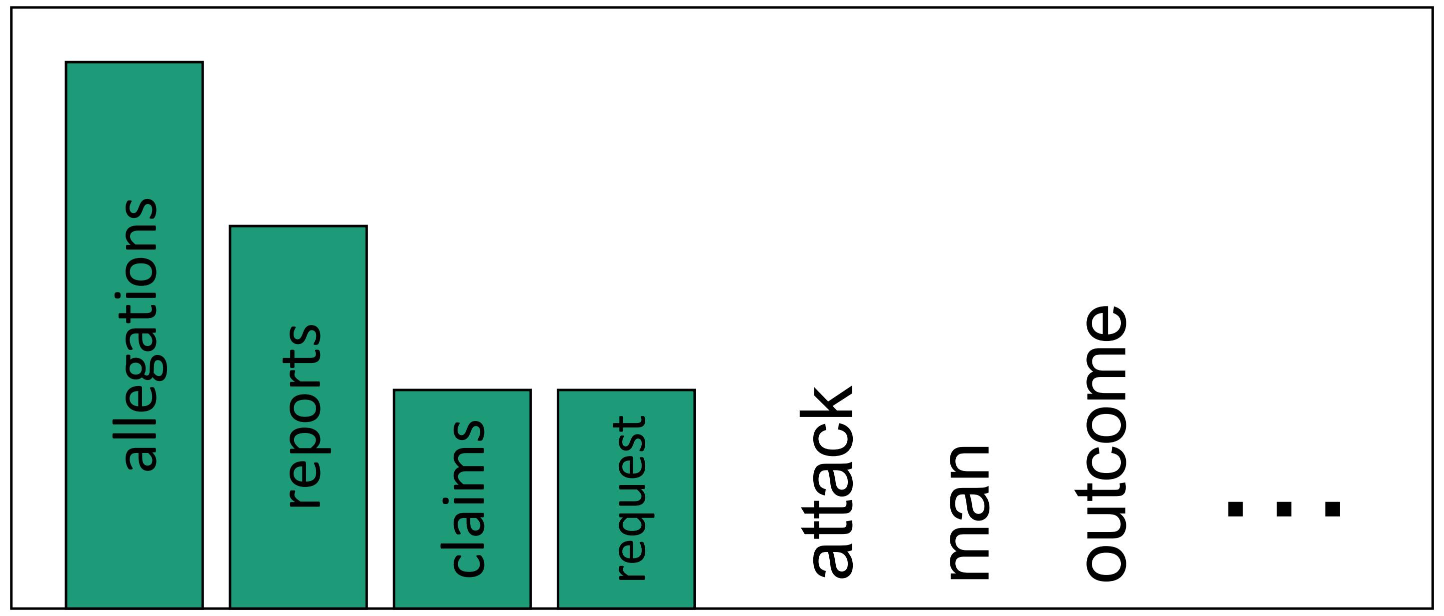
In *Laplace smoothing* or *add-one smoothing*, we pretend we saw each word one more time than we did.

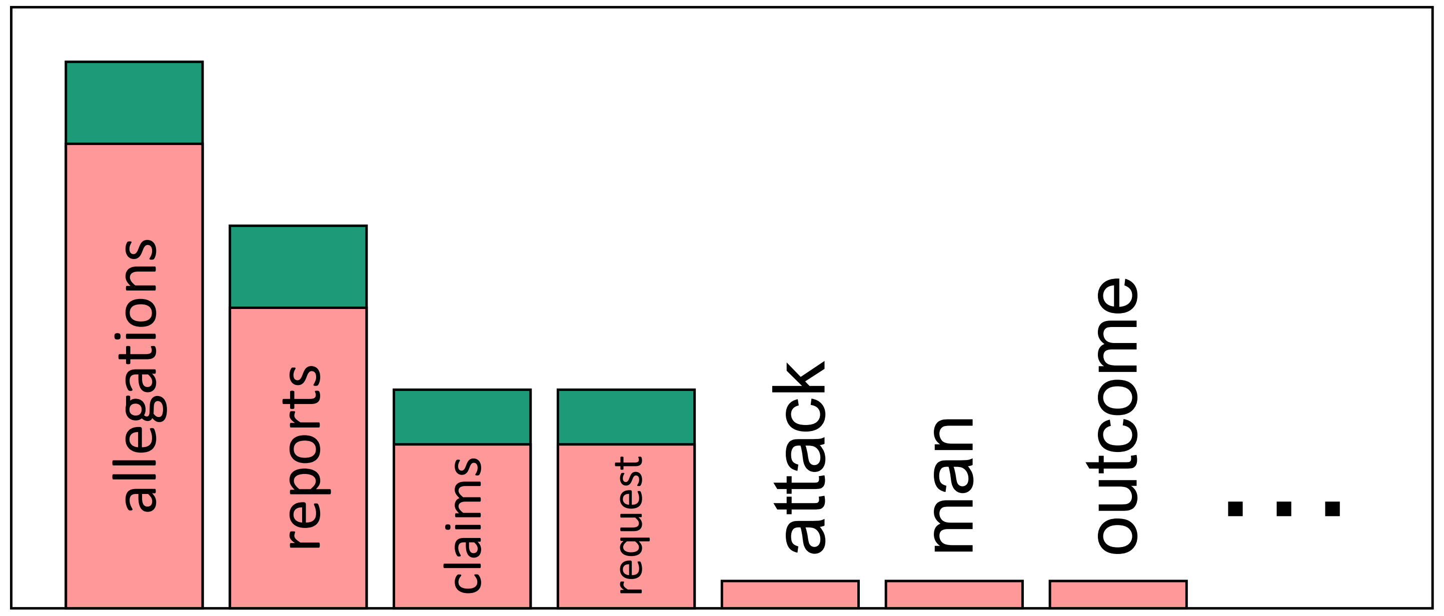




$$P(w_i \mid w_{i-1}, w_{i-2}) = \frac{c(w_{i-2}, w_{i-1}, w_i)}{c(w_{i-2}, w_{i-1})} \quad \longrightarrow \quad P^*(w_i \mid w_{i-1,}, w_{i-2}) = \frac{c(w_{i-2}, w_{i-1}, w_i) + 1}{c(w_{i-2}, w_{i-1}) + V}$$

In *Laplace smoothing* or *add-one smoothing*, we pretend we saw each word one more time than we did.





$$P(w_i \mid w_{i-1}, w_{i-2}) = \frac{c(w_{i-2}, w_{i-1}, w_i)}{c(w_{i-2}, w_{i-1})} \longrightarrow P^*(w_i \mid w_{i-1,}, w_{i-2}) = \frac{c(w_{i-2}, w_{i-1}, w_i) + 1}{c(w_{i-2}, w_{i-1}) + \boxed{V}}$$

Size of the vocabulary

Berkeley Restaurant Project: Original bigram counts

w_2

		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
w_1	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

Berkeley restaurant corpus: Laplace-smoothed bigram counts

w_2

w_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

Berkeley restaurant corpus: Laplace-smoothed bigram probabilities

w_2

w_1

	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

We can go from add-1 smoothing to add-*k*, letting us adjust how smooth the resulting distribution is.

Laplace smoothing is a blunt instrument.

It isn't usually used for *n*-grams; there are better methods.

However, it is used to smooth other NLP models, e.g.,

- for text classification or
- in domains where the number of zeros isn't so huge.

Interpolation

Sometimes it helps to use *less* context.

Condition on less context for contexts you haven't learned much about

Backoff:

Use trigram if you have good evidence.

Otherwise, use bigram.

Otherwise, use unigram.

Interpolation

Mix unigrams, bigrams, and trigrams.

Works better than backoff!

Linear interpolation

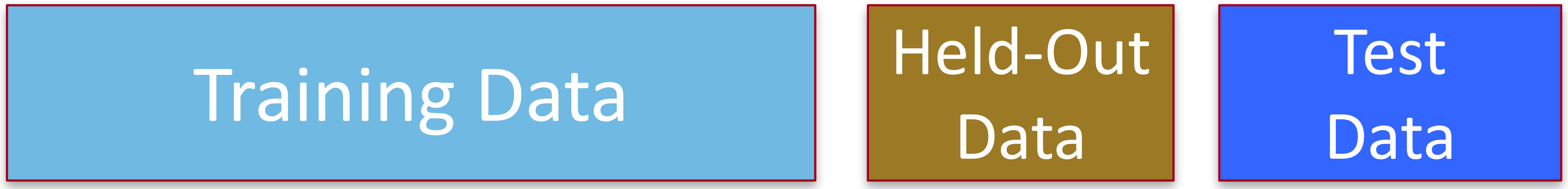
Simple interpolation: Estimate the trigram probabilities by mixing unigram, bigram, and trigram probabilities:

$$\hat{P}(w_n \mid w_{n-2}w_{n-1}) = \lambda_1 P(w_n) + \lambda_2 P(w_n \mid w_{n-1}) + \lambda_3 P(w_n \mid w_{n-2}w_{n-1})$$

where the λs sum to 1

How do we pick the best λ values?

Use a *held-out* corpus:



Choose λs to maximize the probability of held-out data:

Fix the n-gram probabilities (on the training data)

Then search for λs that give the largest probability to the held-out set:

$$\log P(w_1 \ldots w_n \mid M(\lambda_1 \ldots \lambda_k)) = \sum_i \log P_{M(\lambda_1 \ldots \lambda_k)}(w_i \mid w_{i-1})$$

Unknown words

If we know all the words in advance, it's a *closed vocabulary* task.

In an *open vocabulary* task, we might see words at test time that we didn't encounter in training.

These are called *out of vocabulary* (OOV) words.

We define the token `<UNK>` for unknown words.

Training of `<UNK>` probabilities:

- Create a fixed lexicon *L* of size *V*.
- At the text normalization phase, any training word not in *L* changed to `<UNK>`.
- Train language model probabilities as if `<UNK>` were a normal word.

At decoding time,

- Use `<UNK>` probabilities for any word not in training.

Evaluating language models

So, we've counted a bunch of words, but is our language model any good?

Does our language model prefer good sentences to bad ones?

Assign higher probability to “real” or frequently observed sentences vs “ungrammatical” or rarely observed ones?

We learn the model from a ***training set*** and test its performance on a ***test set*** of data we haven’t seen.

An ***evaluation metric*** tells us how well our model does on the test set.

Ethics alert!

Does our language model prefer good sentences to bad ones?

Assign higher probability to "real" or frequently observed sentences vs "ungrammatical" or rarely observed ones?

We learn the model from a *training set* and test its performance on a *test set* of data we haven't seen.

An *evaluation metric* tells us how well our model does on the test set.

Beware

We can't allow test sentences into the training set or we'll assign them artificially high probability when we see it in the test set.

This is called ***training on the test set***, and it's bad science.

Extrinsic evaluation of *n*-gram models

The best evaluation for comparing two language models is to use each model in some “real” task like spelling correction, speech recognition, or machine translation.

Run the task and get the accuracy with each model:

- How many misspelled words were corrected properly?
- How many words were translated correctly?

Compare the accuracy with the models.

Running an *extrinsic* (“in vivo”) evaluation can be very time-consuming.

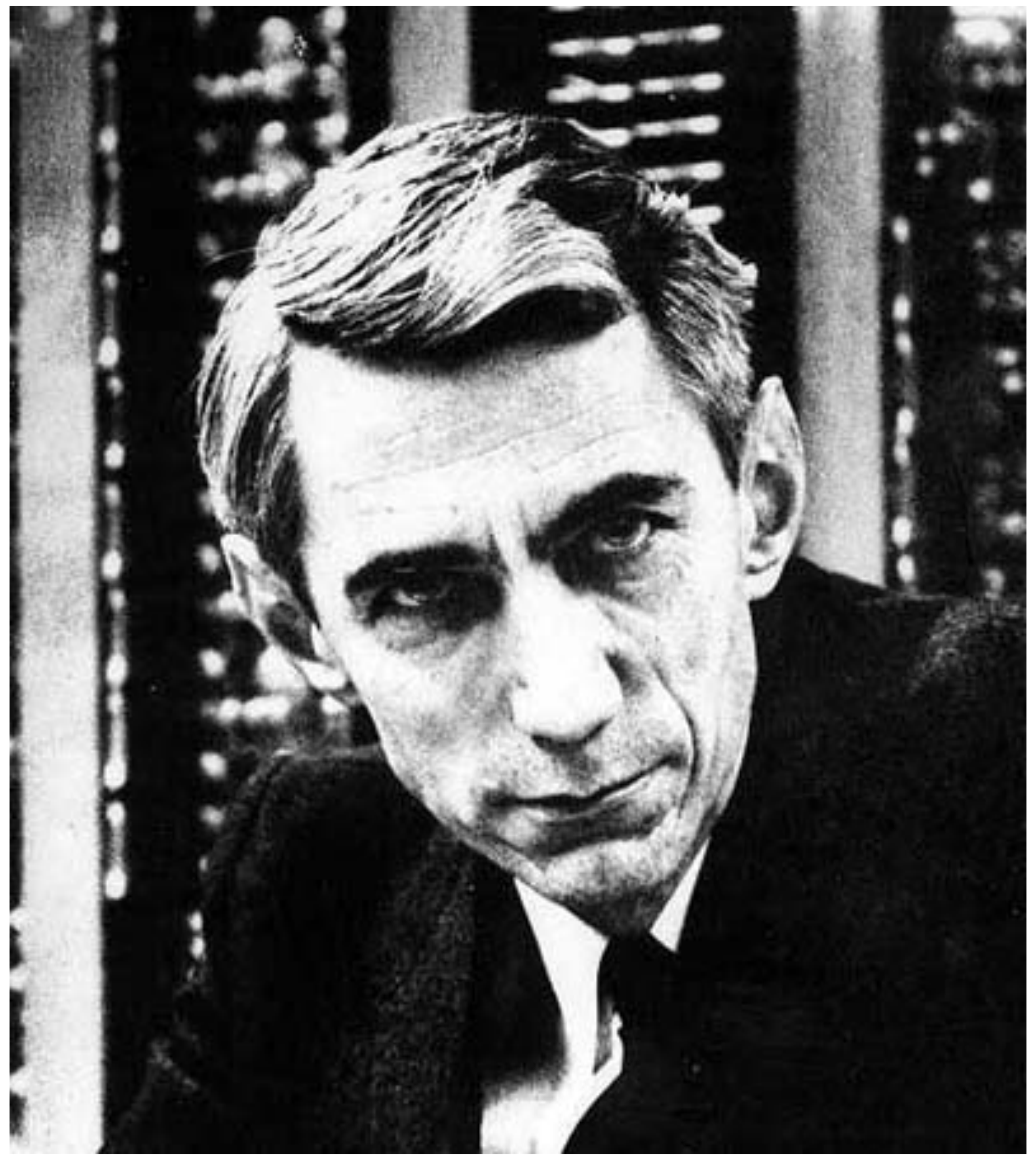
So, sometimes we use an *intrinsic* evaluation like *perplexity*, which is a measure of probability distribution similarity.

Perplexity is a bad approximation unless the test data looks just like the training data.

So, it is generally only used in pilot experiments or to compare models on the same dataset.

Perplexity: Intuition

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



Claude Shannon, looking playful

Perplexity: Intuition

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 ______

mushrooms	0.1
pepperoni	0.1
anchovies	0.01
…	
fried rice	0.0001
…	
and	1e-100

Perplexity: Intuition

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 _____

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 higher probability to the word that *actually* occurs

Unigrams are terrible at this game.

Perplexity

The best language model is one that best predicts an unseen test set – gives the highest probability to the sentences.

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

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

Perplexity

The best language model is one that best predicts an unseen test set – gives the highest probability to the sentences.

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

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}} = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i \mid w_1 \ldots w_{i-1})}} = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i \mid w_{i-1})}}$$

chain rule *for bigrams*

You can think of perplexity as a measure of how "surprised" a model is by some data.

- Low perplexity means the data is highly probable under the model; its not surprised to see it.

Minimizing perplexity is the same as maximizing probability.

- The lower the perplexity, the better the model.

Example: Wall Street Journal corpus

Training set: 38 million words

Test set: 1.5 million words

Perplexity:

Unigram: 962

Bigram: 170

Trigram: 109 ← *Lowest perplexity; best model*

Hands-on practice:
SpaCy and n-grams

```python
from spacy.lang.en import English

nlp = English(pipeline=[])
```

This is a pipeline for processing natural language, including performing tokenization.

```python
from spacy.lang.en import English

nlp = English(pipeline=[])

doc = nlp("Hello world!")

for token in doc:
    print(token.text)
```

```
Hello
world
!
```

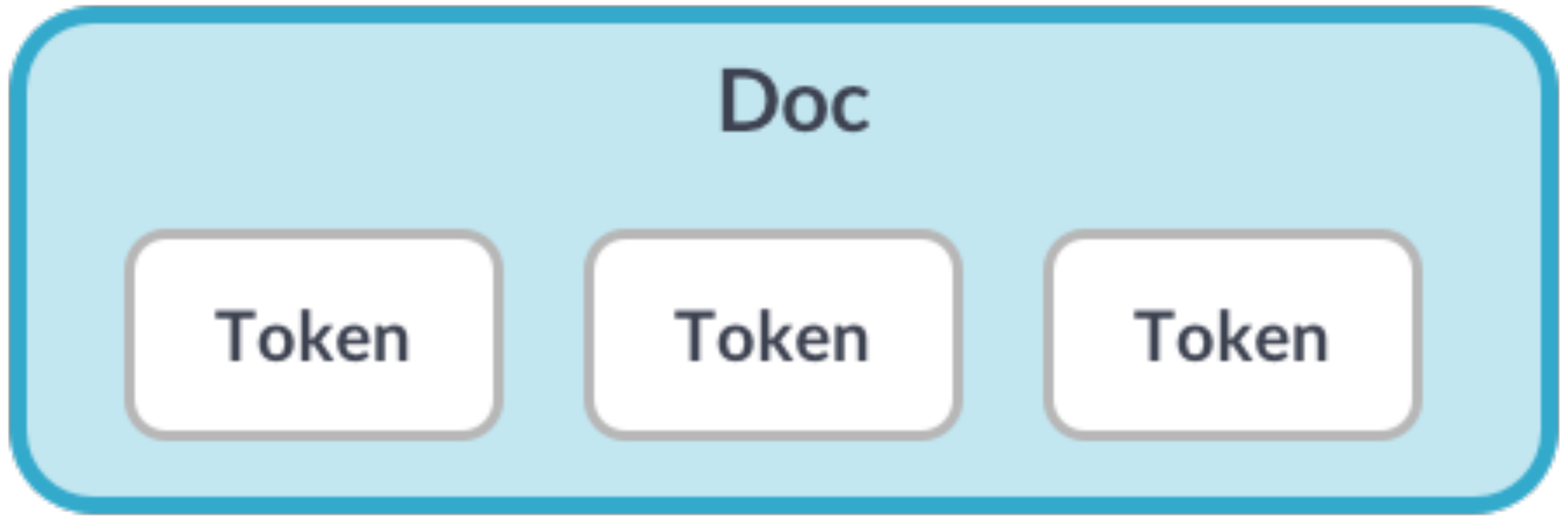
```python
from spacy.lang.en import English

nlp = English(pipeline=[])

doc = nlp("Hello world!")

token = doc[1]
print(token.text)
```

```
world
```



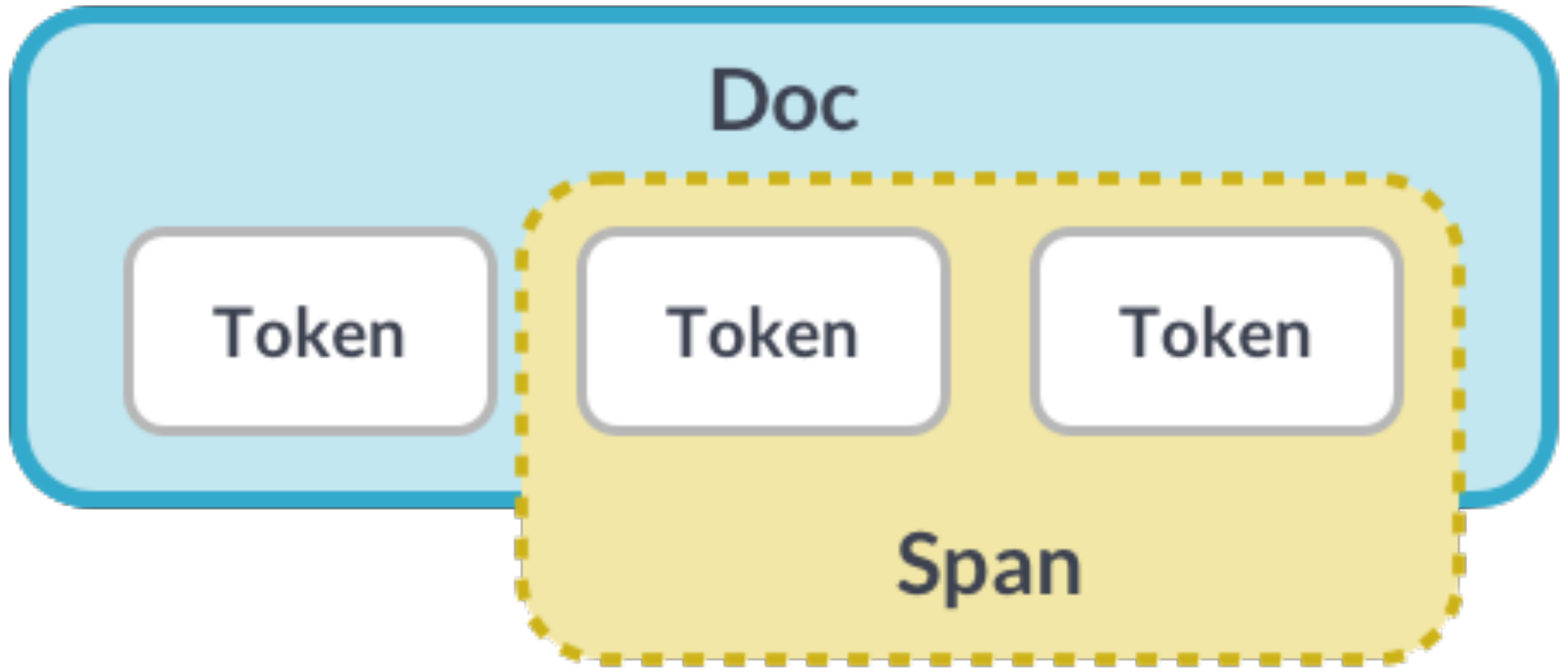
```python
from spacy.lang.en import English

nlp = English(pipeline=[])

doc = nlp("Hello world!")

span = doc[1:3]
print(span.text)
```

```
world!
```

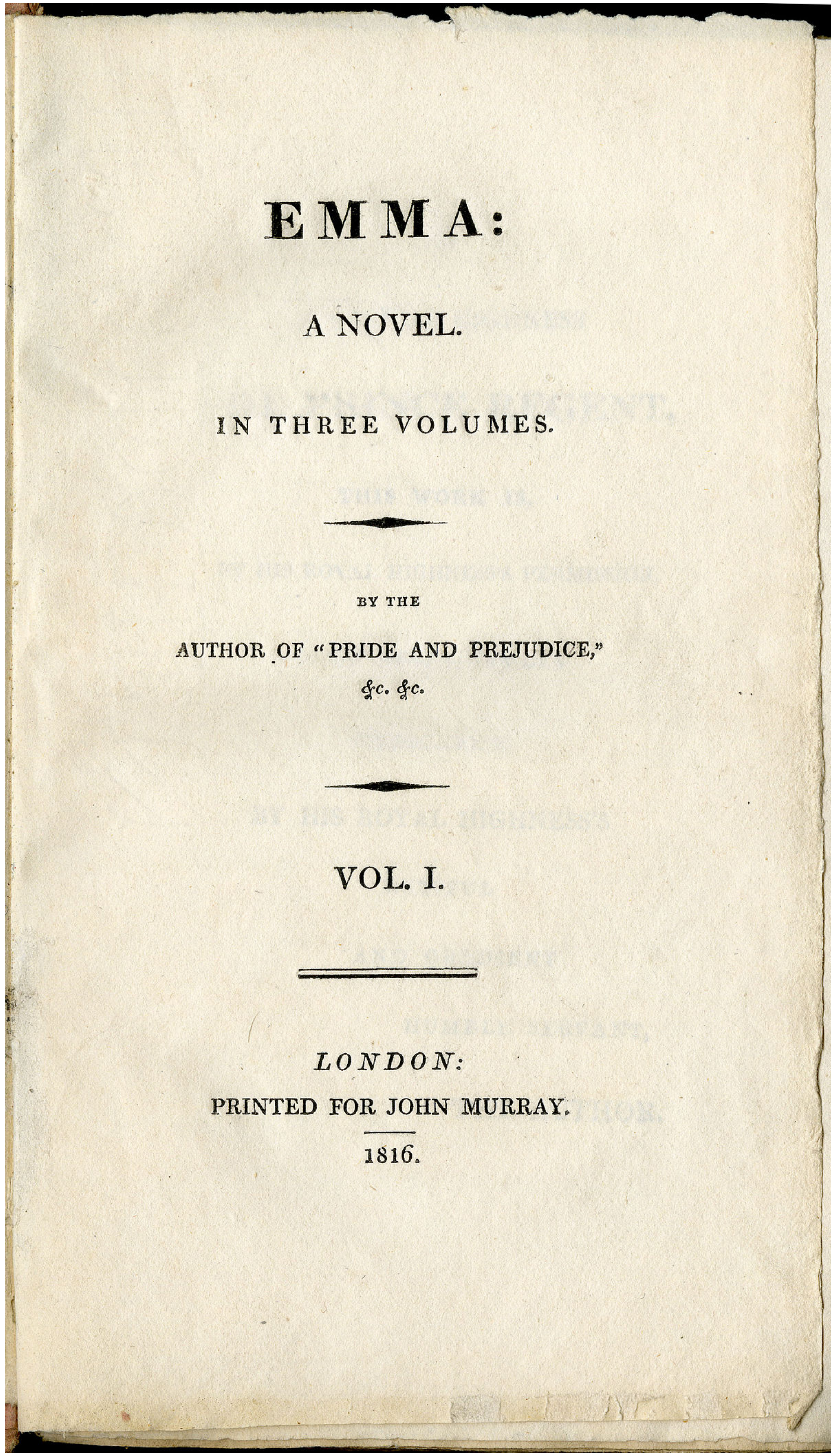


Let's use what we know about working with text in Python – including what we just saw about spaCy – to investigate the frequency of

words (unigrams) and

colocations (bigrams)

in Jane Austen's novel *Emma*.



Acknowledgments

This class incorporates material from:

Carolyn Anderson, Wellesley College

Na-Rae Han, University of Pittsburgh

Nancy Ide, Vassar College

Ines Montani, *Advanced NLP with spaCy*

Jurafsky & Martin, *Speech and Language Processing*, 3rd ed. draft