Introduction to Natural language processing (CSEG321, CSE5321, Spring 2025)

n-Gram Language Models

Instructor:

Hwaran Lee

Assistant Professor

Dept. of Artificial Intelligence

Dept. of Computer Science & Engineering

Sogang University, Seoul, South Korea

Class Objective

Understanding the concept of language models through n-gram models

- What is an n-gram language model?
- Generating from a language model
- Evaluating a language model (perplexity)
- Smoothing (additive, interpolation, discounting)

동해물과 백두산이 __________ ____________

What is a n-gram language model?

What is a n-gram Language Model?

Definition

- A ***probabilistic model*** of a sequence of words
- Joint probability distribution of words $w_1, w_2, \ldots, w_n$:

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

"How likely is a given *phrase, sentence, paragraph* or even a *document?*"

- $P(w_1, w_2, \ldots, w_n)$ associated with every finite word sequence $[w_1, w_2, \ldots, w_n]$

Sample space
= Corpus
= Finite pieces of text

What is a n-gram Language Model?

Chain rule

conditional probability:
$P(w \mid w_1, w_2), w \in V$

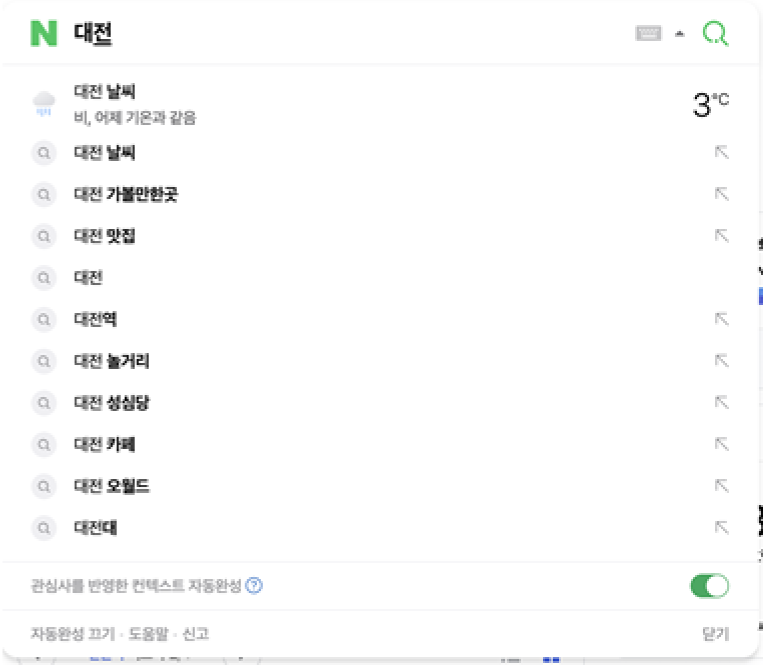
- $P(w_1, w_{,2}, w_3, \dots, w_n)$
  $= P(w_1) * P(w_2|w_1) * P(w_3|w_1, w_2) * P(w_4|w_1, w_2, w_3) * \cdots * P(w_n|w_1, w_2, \dots, w_{n-1})$

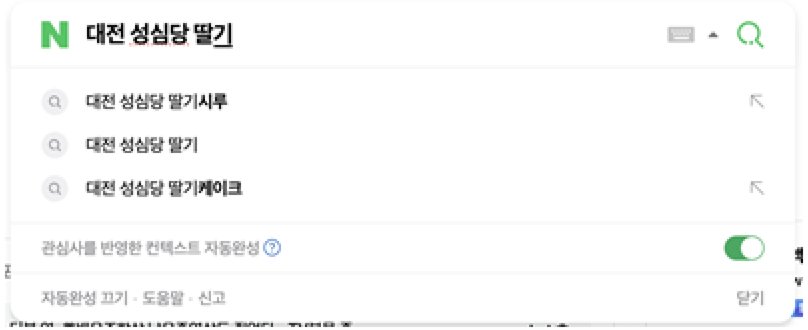
- Example: "the cat sat on the mat"
  P(the cat sat on the mat)
  = P(the) * P(cat | the) * P(sat | the cat) * P(on | the cat sat) * P(the | the cat sat on)
  * P(mat | the cat sat on the)

What is a n-gram Language Model?

Chain rule

- Language models are everywhere





What is a n-gram Language Model?

bigram

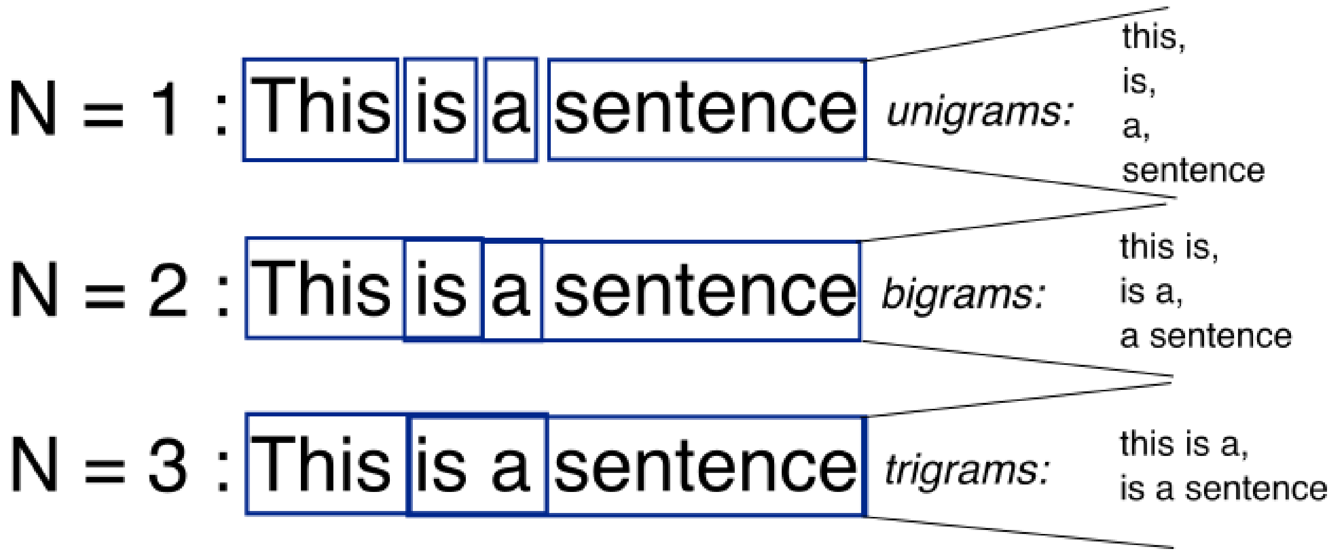
Estimating probabilities

trigram

P(sat | the cat) = count(the cat sat) / count(**the cat**)

P(on | the cat sat) = count(the cat sat on) / count(**the cat sat**)

Maximum Likelihood Estimation (MLE)!



What is a n-gram Language Model?

Estimating probabilities

- Assuming we have a vocabulary of size V,
  how many sequences of length n do we have?

  a) $n * V$
  b) n^V
  c) V^n
  d) V/n

What is a n-gram Language Model?

Estimating probabilities

- Assuming we have a vocabulary of size V,

  how many sequences of length n do we have? $\boldsymbol{V^n}$**!**

- Typical English vocabulary ~ 40k words
  - Even sentences of length <= 11 results in more than $4 * 10^{50}$ sequences.
  - Too many to count! (# of atoms in the earth ~ 10^50)

What is a n-gram Language Model?

Markov assumption

- Use only the recent past to predict the next word
- Reduces the number of estimated parameters in exchange for modeling capacity
- 1st order
  - P(mat | the cat sat on the) ~= P(mat | the)
- 2nd order
  - P(mat | the cat sat on the) ~= P(mat | on the)

Andrey Markov

What is a n-gram Language Model?

K^{th} **order Markov**

- Consider only the last k words (or less) for context

$$P(w_i \mid w_1, w_2, \dots, w_{i-1}) \cong P(w_i \mid w_{i-k}, \dots, w_{i-1})$$

which implies the probability of a sequence is:

$$P(w_1, w_2, \dots, w_n) \cong \prod_i P(w_i \mid w_{i-k}, \dots, w_{i-1})$$

Need to estimate counts for up to (k+1) grams

What is a n-gram Language Model?

n-gram language model

- unigram

$$P(w_1 w_2 w_3 \dots w_n) = \Pi_{i=1}^{n} P(w_i)$$

ex) P(the cat sat on) = P(the) * P(cat) * P(sat) * P(on)

- Trigram

$$P(w_1 w_2 w_3 \dots w_n) = \Pi_{i=1}^{n} P(w_i | w_{i-1})$$

ex) P(the cat sat on) = P(the) * P(cat | the) * P(sat | the cat) * P(on | cat sat)

- and 4-gram, and so on.

Larger the n, more accurate and better the language model, but also higher costs

Generating from a language model

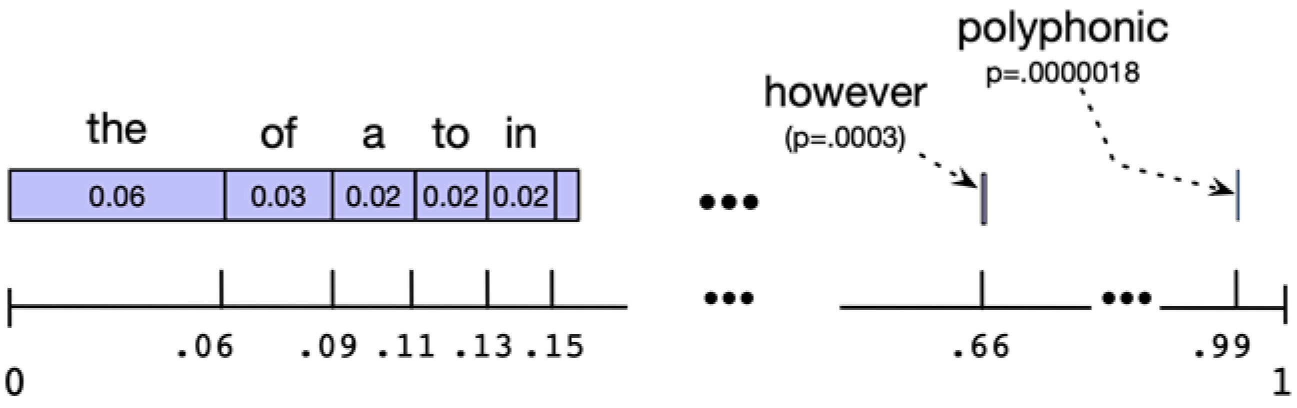
Generating from a Language Model

Generation with n-gram language model

- Given a language model, how to generate a sequence?

  bigram $P(w_1, w_2, \dots, w_n) = \prod_{i=1}^{n} P(w_i|w_{i-1})$

  - Generate the first word $w_1 \sim P(w)$
  - Generate the second word $w_2 \sim P(w\,|w_1)$
  - Generate the third word $w_3 \sim P(w\,|w_2)$
  - …



Generating from a Language Model

Generation with n-gram language model

- Given a language model, how to generate a sequence?

  trigram $P(w_1, w_2, \dots, w_n) = \Pi_{i=1}^{n} P(w_i \mid w_{i-2}, w_{i-1})$

  - Generate the first word $w_1 \sim P(w)$
  - Generate the second word $w_2 \sim P(w \mid w_1)$
  - Generate the third word $w_3 \sim P(w \mid w_1, w_2)$
  - Generate the fourth word $w_4 \sim P(w \mid w_2, w_3)$
  - ...

Generating from a Language Model

Generation results

- unigram

```
release millions See ABC accurate President of
Donald Will cheat them a CNN megynkelly experience @
these word out- the
```

- bigram

```
Thank you believe that @ ABC news, Mississippi
tonight and the false editorial I think the great
people Bill Clinton
```

- trigram

```
We are going to MAKE AMERICA GREAT AGAIN!
#MakeAmericaGreatAgain https: //t.co/DjkdAzT3WV
```

Typical LMs are not sufficient to handle long-range dependencies

Generating from a Language Model

Generation results

- Example of ChatGPT

prompt a.k.a conditional context

Do you know how to write good papers? Writing good papers requires a blend of skills. Firstly, grasp the topic thoroughly; research extensively for credible sources. Next, organize your thoughts logically, crafting a clear thesis statement. Ensure coherence and flow in your writing, supporting arguments with evidence. Pay attention to grammar, punctuation, and style. Finally, revise and edit meticulously for clarity and effectiveness. Practice and feedback refine these skills.

$$P(w_1, w_2, \dots, w_n) = \Pi_{i=1}^{n} P(w_i | w_{i-1024}, \dots, w_{i-2}, w_{i-1})$$

Modern LMs can handle much longer contexts!

Generating from a Language Model

```
>>> # With `top_k` sampling, the output gets restricted the k most likel
>>> # Pro tip: In practice, LLMs use `top_k` in the 5-50 range.
>>> outputs = model.generate(**inputs, do_sample=True, top_k=2)
```

Generation methods

- Greedy: choose the most likely word
  - To predict the next word given a context of two words w_1, w_2:

$$w_3 = argmax_{w \in V} P(w \mid w_1, w_2)$$

- Top-k vs top-p sampling (also called nucleus sampling):

Token	Probability score
for	0.4
to	0.25
with	0.17
and	0.13
by	0.05

Token	Probability score
for	0.4 + 0.25 = 0.65
to	
with	0.17
and	0.13
by	0.05

Evaluating a language model

Evaluating a Language Model

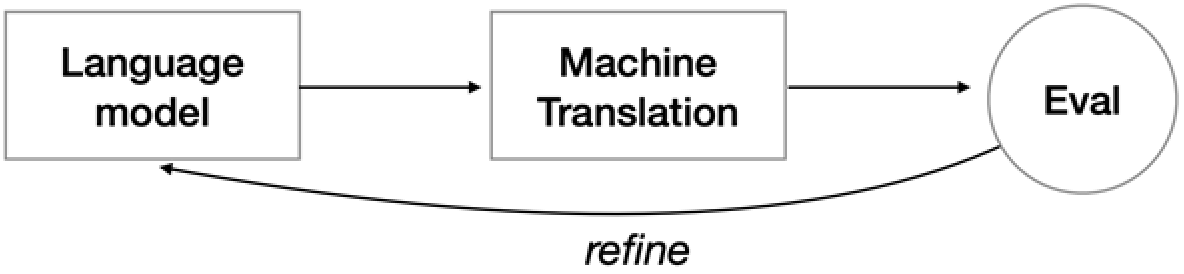
Extrinsic evaluation

- Train LM apply to task observe accuracy

- Directly optimized for downstream applications
  - Higher task accuracy -> better model

- Expensive, time consuming

- Hard to optimize downstream objective (indirect feedback)



Evaluating a Language Model

Intrinsic evaluation of language models

- Research process:
  - Train parameters on a suitable training corpus
    - Assumption: observed sentences ~ good sentences
  - Test on different, unseen corpus
    - If a language model assigns a higher probability to the test set, it is better
  - Evaluation metric - perplexity

Evaluating a Language Model

Perplexity (ppl)

- Measure of how well a LM predicts the next word
  - For a test corpus with words $w_1, w_2, \ldots, w_n$

$$Perplexity = P(w_1, w_2, \ldots, w_n)^{-\frac{1}{n}}$$

$$ppl(S) = e^x \ where \ x = -\frac{1}{n}\log P(w_1, \ldots, w_n) = -\frac{1}{n}\sum_{i=1}^{n}\log P(w_i \mid w_1 \ldots w_{i-1})$$

Cross-entropy

- Unigram model: $x = -\frac{1}{n}\sum_{i=1}^{n}\log P(w_i)$

- Minimizing perplexity ~ maximizing probability of corpus

Evaluating a Language Model

Intuition on perplexity

- If our k-gram model (with vocabulary V) has following probability:

$$P(w_i|w_{i-k+1}, \dots, w_{i-1}) = \frac{1}{|V|}, \qquad \forall w \in V$$

- what is the perplexity of the test corpus?

$A)\ e^{|V|} \qquad B)\ |V| \qquad C)\ |V|^2 \qquad D)\ e^{-|V|}$

$$ppl(S) = e^x \ where\ x = -\frac{1}{n}\log P(w_1, \dots, w_n) = -\frac{1}{n}\sum_{i=1}^{n}\log P(w_i\,|w_1 \dots w_{i-1})$$

Cross-entropy

Evaluating a Language Model

Intuition on perplexity

- If our k-gram model (with vocabulary V) has following probability:

$$P(w_i|w_{i-k+1},\ldots,w_{i-1}) = \frac{1}{|V|}, \qquad \forall w \in V$$

- what is the perplexity of the test corpus?

$A)\ e^{|V|} \qquad B)\ |V| \qquad C)\ |V|^2 \qquad D)\ e^{-|V|}$

$$ppl(S) = e^x \text{ where } x = -\frac{1}{n}\sum_{i=1}^{n} \log P(w_i\,|w_1 \ldots w_{i-1})$$

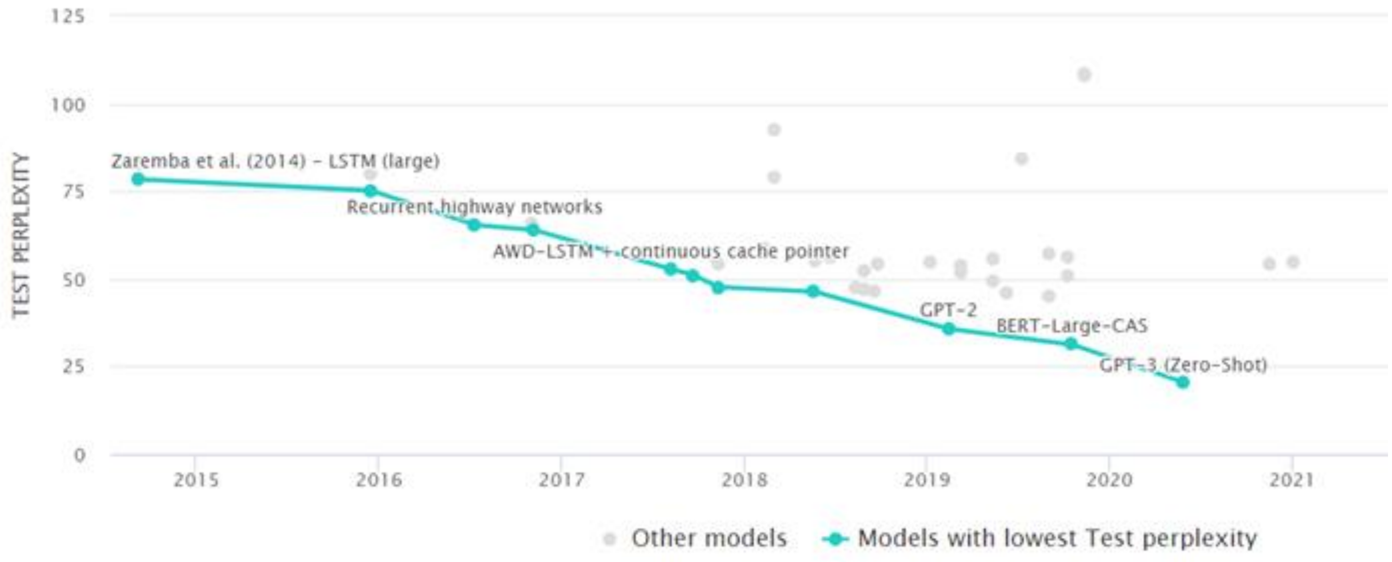
$$ppl(S) = e^{-\frac{1}{n} n \log(\frac{1}{|V|})} = |V|$$

Evaluating a Language Model

Perplexity

- Training corpus 38 million words, test corpus 1.5 million words, both WSJ

n-gram	unigram	bigram	trigram
perplexity	962	170	109



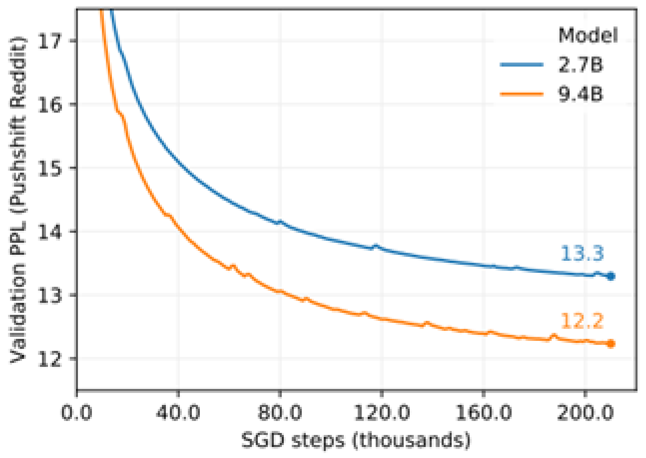


Figure 5: Validation PPL of different sized models. The larger model achieves a better performance in fewer steps, consistent with other works (Kaplan et al., 2020; Li et al., 2020).

Smoothing

Smoothing

Generalization of n-grams

- Any problems with n-gram models and their evaluation?

- Not all n-grams in the test set will be observed in training data

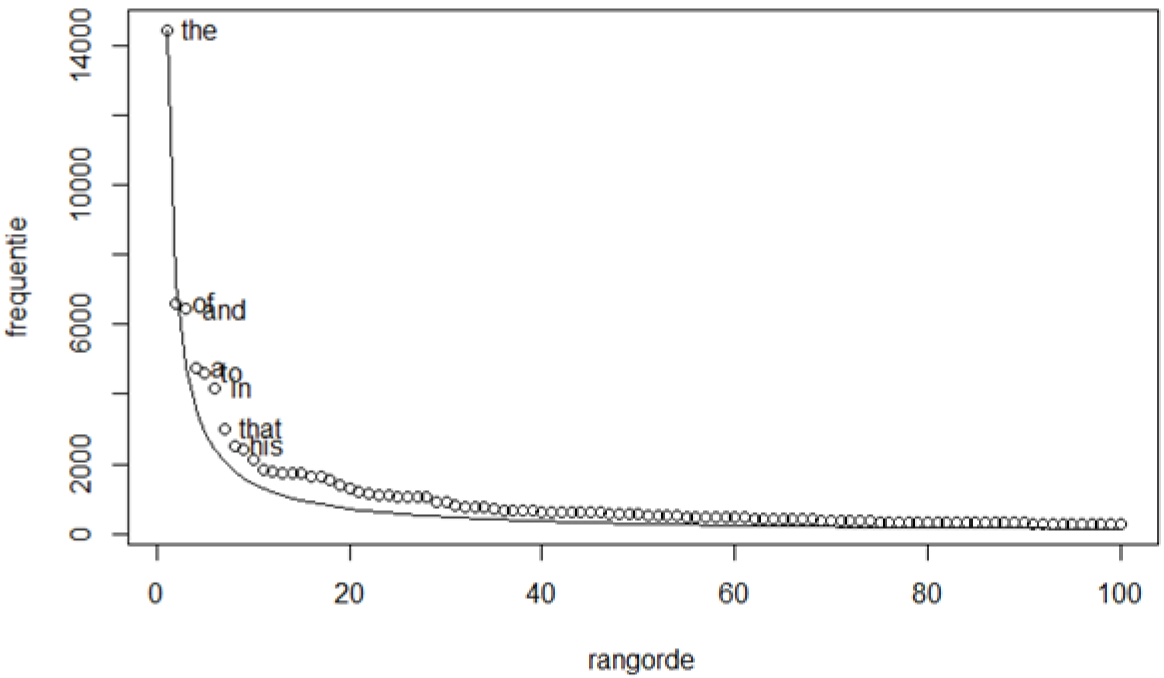
- Test corpus might have some that have zero probability
  - Training set: Google news
  - Test set: Shakespeare

- P(affray | voice doth us) = 0 -> P(test corpus) = 0

- Perplexity is not defined

Smoothing

Sparsity in language

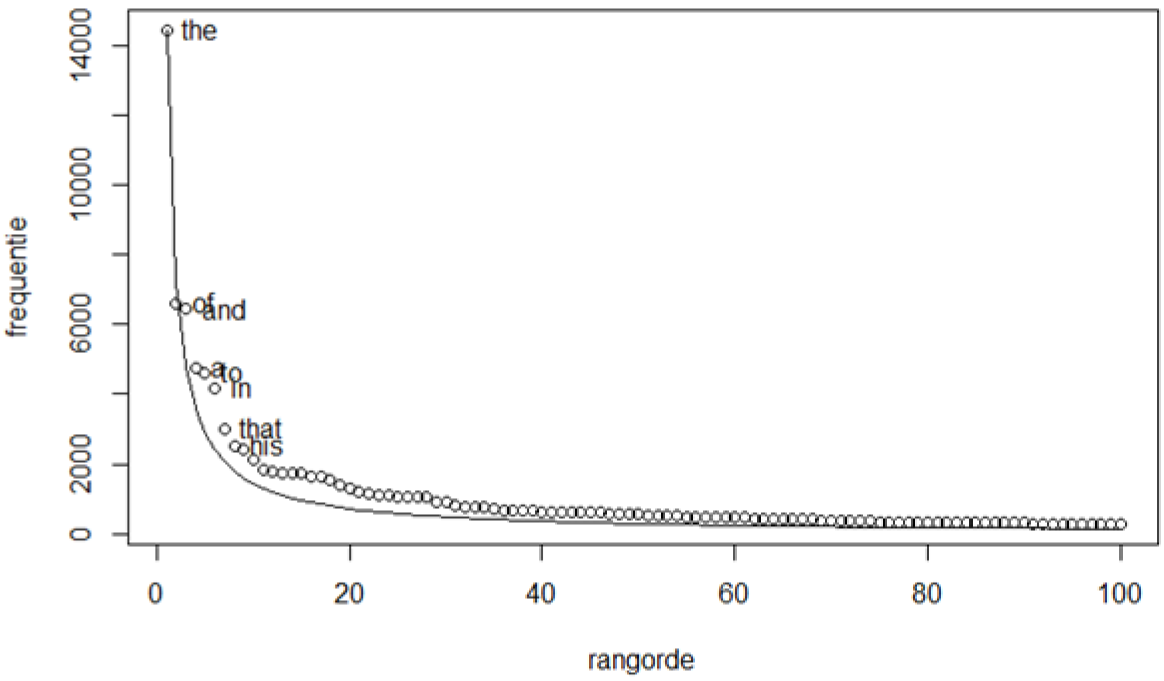


freqeuncy $\propto \frac{1}{rank}$
Zipf's Law

- Long tail of infrequent words
- Most finite-size corpora will have this problem.

Smoothing

Sparsity in language



$$\text{freqeuncy} \propto \frac{1}{rank}$$

Zipf's Law

- Long tail of infrequent words
- Most finite-size corpora will have this problem.

Smoothing

Concept of Smoothing

- Handle sparsity by making sure all probabilities are non-zero in our model
  - **Additive**: Add a small amount to all probabilities
  - **Interpolation**: Use a combination of different granularities of n-grams
  - **Discounting**: Redistribute probability mass from observed n-grams to unobserved ones

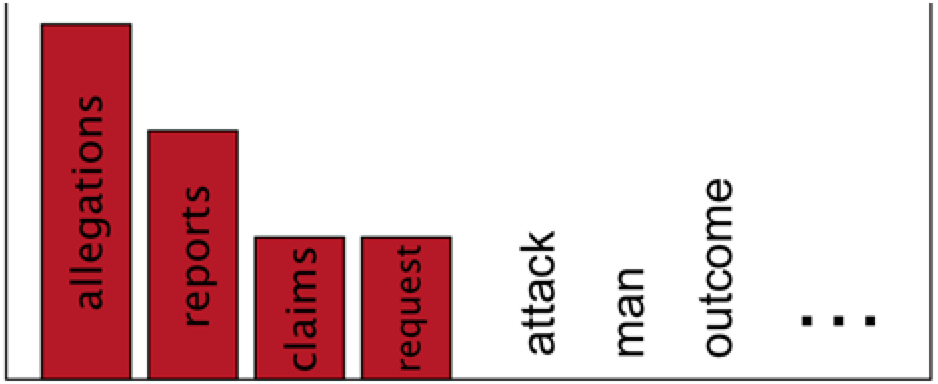
Smoothing

Smoothing intuition

- When we have sparse statistics:

  P(w | denied the)
  3 allegations
  2 reports
  1 claims
  1 request

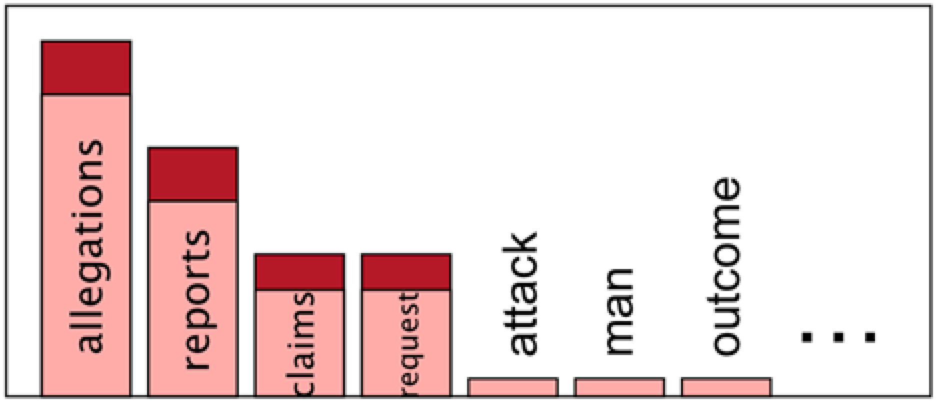
  7 total



- 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



Smoothing

Laplace smoothing

- Also known as add-alpha

- Simplest for of smoothing: Just add α to all counts and renormalize!

- Max likelihood estimate for bigrams:

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

- After smoothing:

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + \alpha}{C(w_{i-1}) + \alpha|V|}$$

Smoothing

Ray bigram counts (Berkeley restaurant corpus)

- Out of 9,222 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

Smoothing

Smoothed bigram counts (Berkeley restaurant corpus)

- Out of 9,222 sentences

	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

Smoothing

Smoothed bigram probabilities (Berkeley restaurant corpus)

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + \alpha}{C(w_{i-1}) + \alpha|V|}, \qquad \alpha = 1$$

	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

Smoothing

Linear interpolation

$$\hat{P}(w_i|w_{i-2}, w_{i-1}) = \lambda_1 P(w_i \mid w_{i-2}, w_{i-1}) + \lambda_2 P(w_i \mid w_{i-1}) + \lambda_3 P(w_i)$$

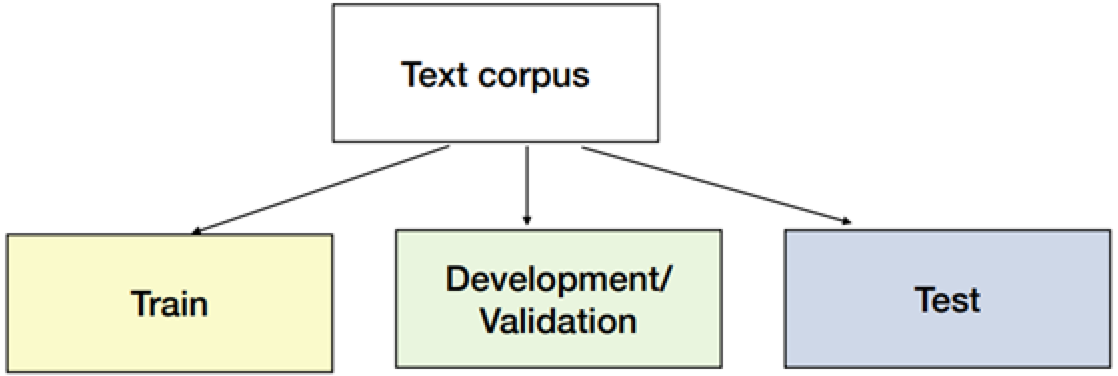
$$\sum_i \lambda_i = 1$$

- Use a combination of models to estimate probability

- Strong empirical performance

Smoothing

Linear interpolation – How can we choose lambdas?



- First, estimate n-gram probabilities on training set
- Then, estimate lambdas (as hyperparameters) to maximize probability on the held-out development/validation set
- Use best model from above to evaluate on test set

Smoothing

Discounting

- Determine some “mass” to remove from probability estimates

- More explicit method for redistributing mass among unseen n-grams

- Just choose an absolute value to discount (usually < 1)

Smoothing

Absolute Discounting

- Define Count*(x) = Count(x) – 0.5

- Missing probability mass:

$$\alpha(w_{i-1}) = 1 - \sum_{w} \frac{Count^*(w_{i-1}, w)}{Count(w_{i-1})}$$

$$\alpha(the) = 10 \times \frac{0.5}{48} = \frac{5}{48}$$

- Divide this mass between words w for which Count(the, w)=0

x	Count(x)	Count$^*(x)$	$\frac{\text{Count}^*(x)}{\text{Count}(x)}$
the	48		
the, dog	15	14.5	14.5/48
the, woman	11	10.5	10.5/48
the, man	10	9.5	9.5/48
the, park	5	4.5	4.5/48
the, job	2	1.5	1.5/48
the, telescope	1	0.5	0.5/48
the, manual	1	0.5	0.5/48
the, afternoon	1	0.5	0.5/48
the, country	1	0.5	0.5/48
the, street	1	0.5	0.5/48

Smoothing

Absolute Discounting

$$\alpha(the) = 10 \times \frac{0.5}{48} = \frac{5}{48}$$

$$P_{abs-discount}(w_i \mid w_{i-1})$$

$$= \begin{cases} \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})}, if\ c(w_{i-1}, w_i) > 0 \\ \alpha(w_{i-1}) \cdot \frac{P(w_i)}{\sum_{w'} P(w')}, if\ c(w_{i-1}, w_i) = 0 \end{cases}$$

x	$\text{Count}(x)$	$\text{Count}^*(x)$	$\frac{\text{Count}^*(x)}{\text{Count}(x)}$
the	48		
the, dog	15	14.5	14.5/48
the, woman	11	10.5	10.5/48
the, man	10	9.5	9.5/48
the, park	5	4.5	4.5/48
the, job	2	1.5	1.5/48
the, telescope	1	0.5	0.5/48
the, manual	1	0.5	0.5/48
the, afternoon	1	0.5	0.5/48
the, country	1	0.5	0.5/48
the, street	1	0.5	0.5/48

E.O.D