Natural Language Processing

Lecture 3: n-gram language models

Probability of a Sentence

- What is the probability that the Naive Bayes' model actually computes?

"But it must be recognized that the notion of 'probability of a sentence' is an entirely useless one, under any known interpretation of this term."
Noam Chomsky (1969)

Language Modeling

- Task: predict the next word given the context.
- Used in speech recognition, handwritten character recognition, spelling correction, text entry UI, machine translation,…

Language Modeling

- Stocks plunged this …
- Let’s meet in Times …
- I took the subway to …

From a NYT story

- *Stocks plunged this ....*
  - *Stocks plunged this morning, despite a cut interest rates by the …*
  - *Stocks plunged this morning, despite a cut in interest rates by the Federal Reserve, as Wall …*
  - *Stocks plunged this morning, despite a cut in interest rates by the Federal Reserve, as Wall Street began*

Human Word Prediction

- Clearly at least some of us have the ability to predict the future.
- How does this work?
  - Domain knowledge
  - Syntactic knowledge (guess correct part of speech)
  - Lexical knowledge

Probability of the Next Word

- Idea: We do not need to model domain, syntactic, and lexical knowledge perfectly.

- Instead, we can rely on the notion of **probability of a sequence** (letters, words…).

$$P(w_n|w_1, w_2, w_3, \ldots, w_{n-1})$$

Applications

- Speech recognition: P(*“recognize speech”*) > P(*“wreck a nice beach”*)
- Text generation: P(*“three houses”*) > P(*“three house”*)
- Spelling correction P(*“my cat eats fish”*) > P(*“my xat eats fish”*)
- Machine Translation P(*“the blue house”*) > P(*“the house blue”*)
- Other uses
  - OCR
  - Summarization
  - Document classification
  - Essay scoring

Language Models

- This model can also be used to describe the probability of an entire sentence, not just the last word.
- Use the chain rule:

$$\boldsymbol{P(w_1, \dots., w_n) =}$$

$$\boldsymbol{P(w_n | w_1, \dots \dots, w_{n-1})\ P(w_1, \dots., w_{n-1})} =$$

$$\boldsymbol{P(w_n | w_1, \dots., w_{n-1})\ P(w_{n-1} |\ w_{n-2}, \dots., w_1) P(w_{n-2}, \dots., w_1)}$$

$$\dots$$

Markov Assumption

- $P(w_n|w_1, w_2, w_3, \ldots, w_{n-1})$ is difficult to estimate.

- The longer the sequence becomes, the less likely $w_1\ w_2\ w_3 \ldots w_{n-1}$ will appear in training data.

- Instead, we make the following simple independence assumption (Markov assumption):

  - The probability to see w_n depends only on the previous k-1 words.

$$P(w_n|w_1, w_2, w_3, \ldots, w_{n-1})$$

$$\approx P(w_n|w_{n-k+1}, \ldots, w_{n-1})$$

bi-gram language model

- Using the Markov assumption and the chain rule:

$$P(w_1, \ldots, w_n) \approx$$

$$P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_2) \cdots P(w_n|w_{n-1})$$

- More consistent to use only bigrams:

$$P(w_1|start) \cdot P(w_2|w_1) \cdot P(w_3|w_2) \cdots P(w_n|w_{n-1})$$

n-grams

- The sequence w_n is a unigram.
- The sequence w_{n-1}, w_n is a bigram.
- The sequence w_{n-2}, w_{n-1}, w_n is a trigram….
- The sequence w_{n-2}, w_{n-1}, w_n is a quadrigram…

Variable-Length Language Models

- We typically don't know what the length of the sentence is.
- Instead, we use a special marker STOP that indicates the end of a sentence.
- We typically just augment the sentence with START and STOP markers to provide the appropriate context.

START i want to eat Chinese food END

$$P(i|START) \cdot P(want|i) \cdot P(to|want) \cdot P(eat|to) \cdot P(Chinese|eat) \cdot P(food|Chinese) \cdot P(END|food)$$

trigram example

$$P(i|START, START) \cdot P(want|START,i) \cdot P(to|i,want) \cdot P(eat|want,to) \cdot$$
$$P(Chinese|to,eat) \cdot P(food|eat,Chinese) \cdot P(END|Chinese,food)$$

Bigram example from the Berkeley Restaurant Project (BeRP)

Eat on	0.16	Eat Thai	0.03
Eat some	0.06	Eat breakfast	0.03
Eat lunch	0.06	Eat in	0.02
Eat dinner	0.05	Eat Chinese	0.02
Eat at	0.04	Eat Mexican	0.02
Eat a	0.04	Eat tomorrow	0.01
Eat Indian	0.04	Eat dessert	0.007

http://www1.icsi.berkeley.edu/Speech/berp.html

Bigram example from the Berkeley Restaurant Project (BeRP)

START I	0.25	Want some	0.04
START I’d	0.06	Want Thai	0.01
START Tell	0.04	To eat	0.26
START I’m	0.02	To have	0.14
I want	0.32	To spend	0.09
I would	0.29	To be	0.02
I don’t	0.08	British food	0.60
[illegible]	[illegible]	[illegible]	[illegible]

Bigram example from the Berkeley Restaurant Project (BeRP)

- Assume P(END | food) = 0.2

P(I want to eat British food) =
P(I | START) · P(want | I) · P(to | want) · P(eat | to) ·
P(British | eat) · P(food | British) · P(END | food) =
.25 · .32 · .65 · .26 · .001 · .60 · .2 = .0000016

P(I want to eat Chinese food) =
P(I | START) · P(want | I) · P(to | want) · P(eat | to) ·
P(Chinese | eat) · P(food | Chinese) · P(END | food) =
.25 · .32 · .65 · .26 · **.02 · .60** · .2 = .000032

log probabilities

- Probabilities can become very small (a few orders of magnitude per token).

- We often work with log probabilities in practice.

$$p(w_1 \ldots w_n) = \prod_{i=1}^{n} p(w_i | w_{i-1})$$

$$\log p(w_1 \ldots w_n) = \sum_{i=1}^{n} \log p(w_i | w_{i-1})$$

w_0 = START

What do ngrams capture?

- Probabilities seem to capture *syntactic facts* and *world knowledge.*
- *eat* is often followed by a NP.
- *British* food is not too popular, but *Chinese* is.

Estimating n-gram probabilities

- We can estimate n-gram probabilities using maximum likelihood estimates.

$$p(w|u) = \frac{count(u, w)}{count(u)}$$

- Or for trigrams:

$$p(w|u, v) = \frac{count(u, v, w)}{count(u, v)}$$

Bigram Counts from BeRP

	I	Want	To	Eat	Chinese	Food	lunch
I	8	1087	0	13	0	0	0
Want	3	0	786	0	6	8	6
To	3	0	10	860	3	0	12
Eat	0	0	2	0	19	2	52
Chinese	2	0	0	0	0	120	1
Food	19	0	17	0	0	0	0

Counts to Probabilities

$$P(want|I) = \frac{count(I, want)}{count(I)} = 1087/3437 = 0.32$$

	I	Want	To	Eat	Chinese	Food	lunch
I	8	1087	0	13	0	0	0
Want	3	0	786	0	6	8	6
To	3	0	10	860	3	0	12
Eat	0	0	2	0	19	2	52
Chinese	2	0	0	0	0	120	1
Food	19	0	17	0	0	0	0
Lunch	4	0	0	0	0	1	0

Corpora

- Large digital collections of text or speech. Different languages, domains, modalities. Annotated or un-annontated.
- English:
  - Brown Corpus
  - BNC, ANC
  - Wall Street Journal
  - AP newswire
  - DARPA/NIST text/speech corpora
    (Call Home,ATIS, switchboard, Broadcast News,…)
  - MT: Hansards, Europarl

Google Web 1T 5-gram Corpus

File sizes: approx. 24 GB compressed (gzip'ed) text files

Number of tokens: 1,024,908,267,229

Number of sentences: 95,119,665,584

Number of unigrams: 13,588,391

Number of bigrams: 314,843,401

Number of trigrams: 977,069,902

Number of fourgrams: 1,313,818,354

Number of fivegrams: 1,176,470,663

Google Web 1T 5-gram Corpus

- 3-gram examples:

ceramics collectables collectibles 55
ceramics collectables fine 130
ceramics collected by 52
ceramics collectible pottery 50
ceramics collectibles cooking 45
ceramics collection , 144
ceramics collection . 247
ceramics collection </S> 120
ceramics collection and 43
ceramics collection at 52
ceramics collection is 68
ceramics collection of 76

Google Web 1T 5-gram Corpus

- 4-gram examples:

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
serve as the industrial 52
serve as the industry 607
serve as the info 42
serve as the informal 102

Data sparsity in n-gram models

- Sparsity is a problem all over NLP: Test data contains language phenomena not encountered during training.
- For n-gram models there are two issues:
  - We may not have seen all tokens.
  - We may not have seen all ngrams (even though the individual tokens are known).
    - Token has not been encountered in this context before.

$$P(\text{lunch} \mid \text{I}) = 0.0$$

Unseen Tokens

- Typical approach to unseen tokens:
  - Start with a specific lexicon of known tokens.
  - Replace all tokens in the training and testing corpus that are not in the lexicon with an *UNK* token.
  - Practical approach:
    - Lexicon contains all words that appear more than *k* times in the training corpus.
    - Replace all other tokens with UNK.

Unseen Contexts

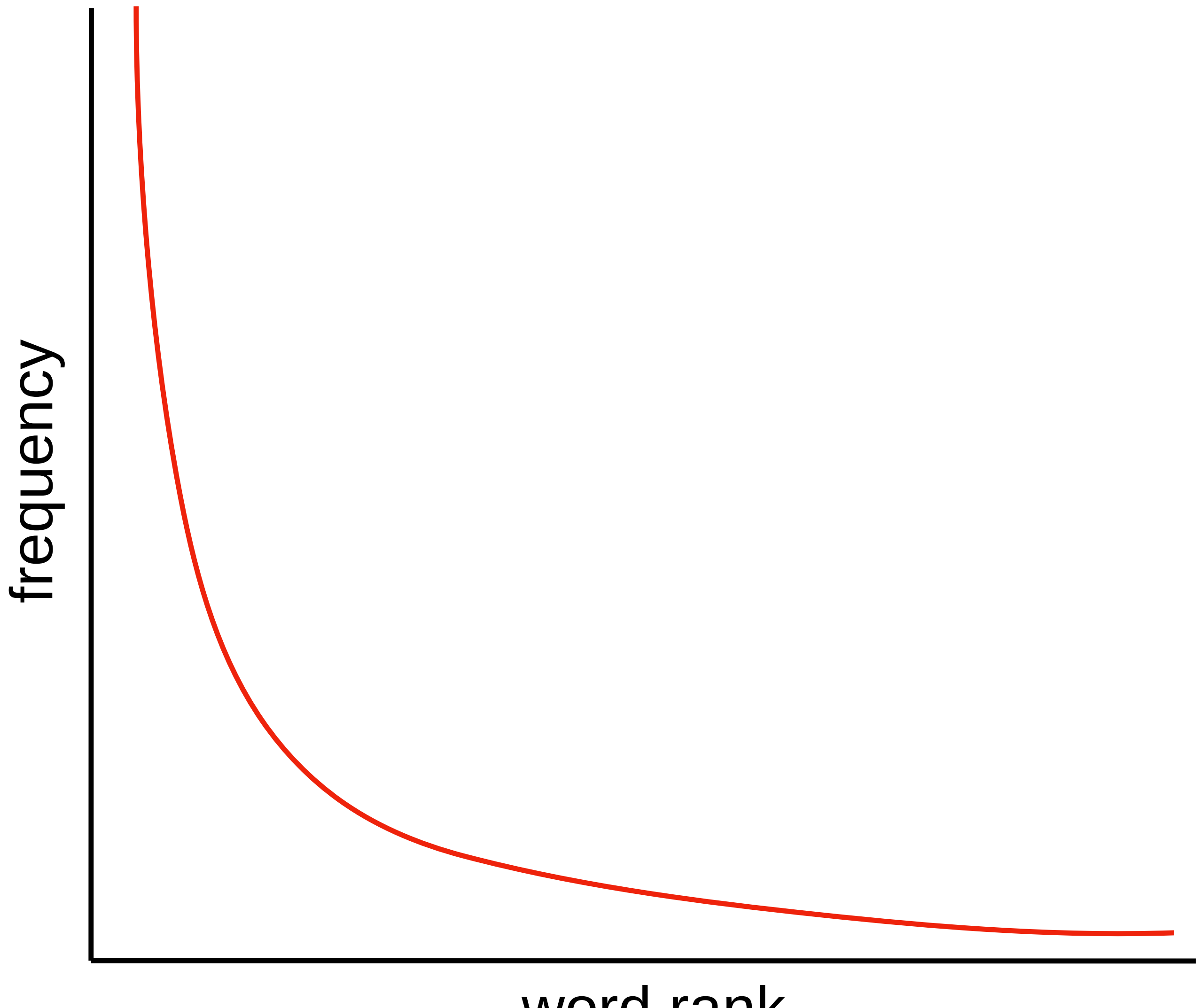
- Two basic approaches:
  - Smoothing / Discounting: Move some probability mass from seen trigrams to unseen trigrams.
  - Back-off: Use n-1-…, n-2-… grams to compute n-gram probability.
- Other techniques:
  - Class-based backoff, use back-off probability for a specific word class / part-of-speech.

Zipf's Law

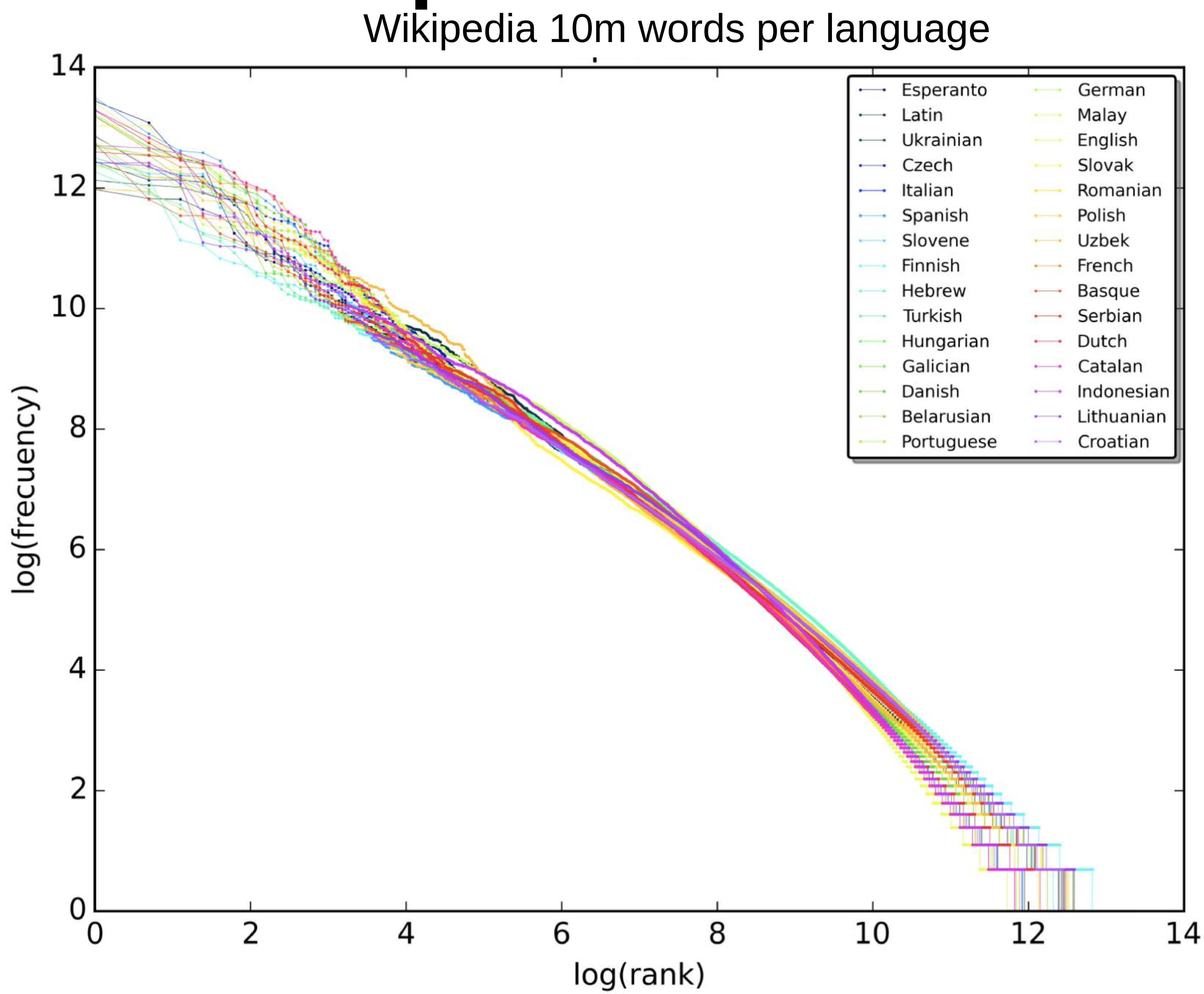
- Problem: n-grams (and most other linguistic phenomena) follow a *Zipfian* distribution.
- A few words occur very frequently.
- Most words occur very rarely. Many are seen only once.

Zipf's law: a word's frequency is approximately inversely proportional to its rank in the word distribution list.

Zipf's Law



Zipf's Law



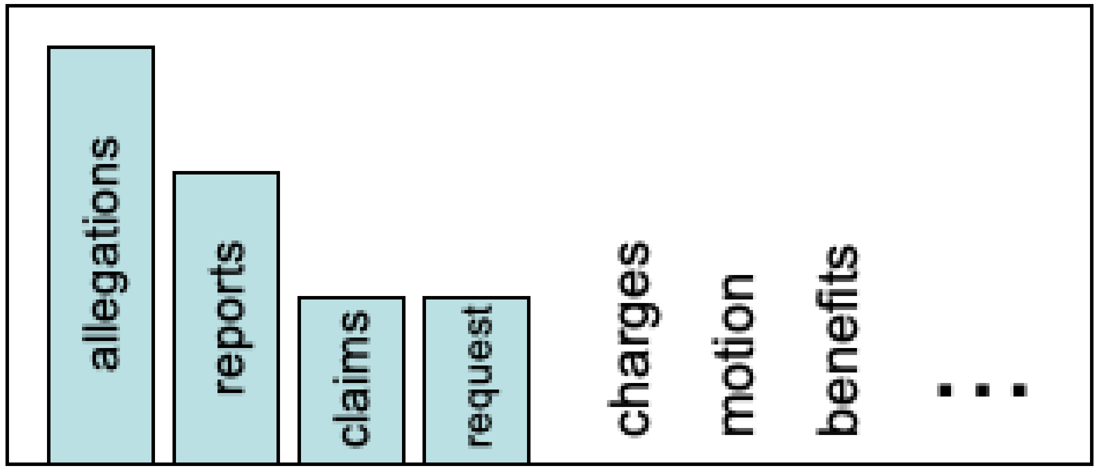
https://commons.wikimedia.org/wiki/File:Zipf_30wiki_en_labels.png

Smoothing

- Smoothing flattens spiky distributions.

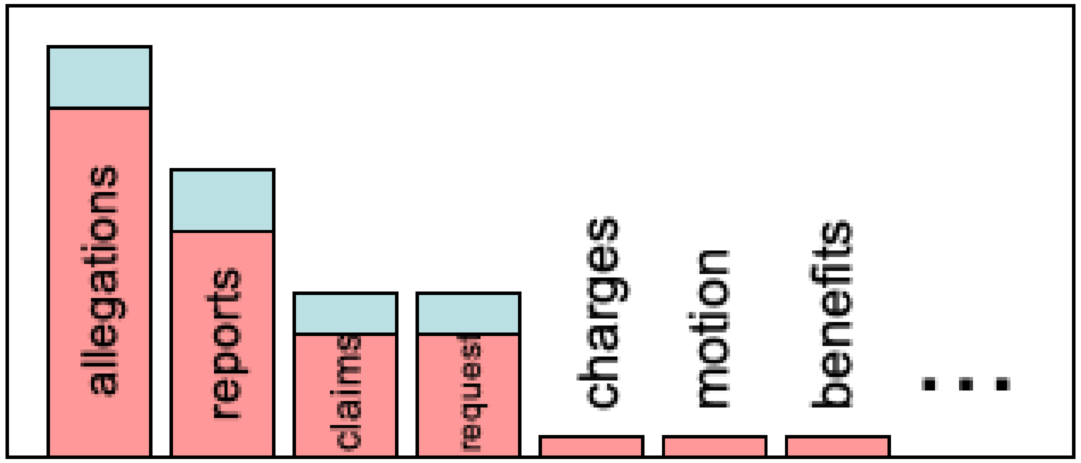
- before P(w | *We denied the)*

  3 allegations
  2 reports
  1 claims
  1 request
  7 total



- after P(w | *We denied the)*

  2.5 allegations
  1.5 reports
  0.5 claims
  0.4 request
  2 UNK
  7 total



Smoothing is like Robin Hood: Steal from the rich, give to the poor.

Example from Dan Klein.

Additive Smoothing

- Classic approach: Laplacian, a.k.a. additive smoothing.

$$P(w_i) = \frac{count(w_i) + 1}{N + V}$$

- N is the number of tokens, V is the number of types (i.e. size of the vocabulary)

$$P(w|u) = \frac{count(u, w) + 1}{count(u) + V}$$

- Inaccurate in practice.

Linear Interpolation

- Use denser distributions of shorter ngrams to “fill in” sparse ngram distributions.

$$p(w|u,v) = \lambda_1 \cdot p_{mle}(w|u,v) + \lambda_2 \cdot p_{mle}(w|v) + \lambda_3 \cdot p_{mle}(w)$$

- Where $\lambda_1, \lambda_2, \lambda_3 > 0$ and $\lambda_1 + \lambda_2 + \lambda_3 = 1$.
- Works well in practice (but not a lot of theoretical justification why).
- Parameters can be estimated on development data (for example, using Expectation Maximization).

Discounting

- Idea: set aside some probability mass, then fill in the missing mass using back-off.
- $count^*(v, w) = count(v, w) - \beta$ where $0 < \beta < 1$.
- Then for all seen bigrams: $p(w|v) = \dfrac{count^*(v, w)}{count(v)}$
- For each context *v* the missing probability mass is

$$\alpha(v) = 1 - \sum_{w:c(v,w)>0} \frac{count^*(v, w)}{count(v)}$$

- We can now divide this held-out mass between the unseen words (evenly or using back-off).

Katz' Backoff

- Divide the held-out probability mass proportionally to the unigram probability of the unseen words in context *v*.

$$p(w|v) = \begin{cases} \frac{count^*(v,w)}{count(v)} & \text{if } count(v,w) > 0 \\ \alpha(v) \times \frac{p_{mle}(w)}{\sum\limits_{u:count(v,u)=0} p_{mle}(u)} & \text{otherwise.} \end{cases}$$

Katz' Backoff for Trigrams

- For trigrams: recursively compute backoff-probability for unseen bigrams. Then distribute the held-out probability mass proportionally to that bigram backoff-probability.

$$p(w|u,v) = \begin{cases} \frac{count^*(u,v,w)}{count(u,v)} & \text{if } count(u,v,w) > 0 \\ \alpha(u,v) \times \frac{p_{BO}(w|v)}{\sum\limits_{z:count(v,z)=0} p_{BO}(z|v)} & \text{otherwise.} \end{cases}$$

- where: $\alpha(u,v) = 1 - \sum\limits_{w:count(u,v,w)>0} \frac{count^*(u,v,w)}{count(u,v)}$

- Often combined with Good-Turing smoothing.

Evaluating n-gram models

- Extrinsic evaluation: Apply the model in an application (for example language classification). Evaluate the application.
- Intrinsic evaluation: measure how well the model approximates unseen language data.
  - Can compute the probability of each sentence according to the model. Higher probability -> better model.
  - Typically we compute *Perplexity instead.*

Perplexity

- Perplexity (per word) measures how well the ngram model predicts the sample.
- Perplexity is defined as 2^{-l} , where $l = \frac{1}{M} \sum_{i=1}^{m} \log_2 p(s_i)$.
- Lower perplexity = better model. Intuition:
  - Assume we are predicting one word at a time.
  - With uniform distribution, all successor words are equally likely. Perplexity is equal to vocabulary size.
  - Perplexity can be thought of as “effective vocabulary size”.