Natural Language Processing

CSCI 5832— Lecture 4

Jim Martin

NLP@CU

Today

More language modeling

- Review random sampling
- Smoothing
  - Laplace smoothing
  - Backoff, interpolation and discounting

Coming Up

- Week 3
  - Tuesday: Part of Speech Tagging (Chapter 8)
  - Thursday: HMMs/Viterbi Alg
- Week 4
  - Tuesday: Text classification/ Naïve Bayes (Chapter 4)
  - Thursday: Text classification/ Logistic Regression (Chapter 5)
- Week 5
  - Tuesday: More logistic regression
  - Thursday: Vector Semantics (Chapter 6) (not on quiz 1)
  - Quiz 1 (Chapters 2, 3, 4, 5, and 8)

Shannon’s Method

- Assigning probabilities to sentences is all well and good, but it’s not terribly illuminating.
- A more entertaining (and very useful) task is to turn the model around and use it to *generate* random sentences that are *similar to* the sentences from which the model was derived.
- Idea generally attributed to

  Claude Shannon.

Shannon's Method (Autoregressive Generation)

- Sample a random bigram (<s>, w_i) according to the model's probability distribution over bigrams beginning with <s>
- Now sample a new random bigram (w_i, x) according to its probability. Where the prefix *w matches the suffix of the first bigram chosen.*
- And so on until we randomly choose a (w_i, </s>)
- Then string them together
- <s> I

  I want

  want to

  to eat

  eat Chinese

  Chinese food

  food </s>

Random Sampling

"One fish two fish red fish blue fish black fish blue fish"

- Assuming a unigram model
  - $N = ?$
  - $|V| = ?$

Random Sampling

"One fish two fish red fish blue fish black fish blue fish"

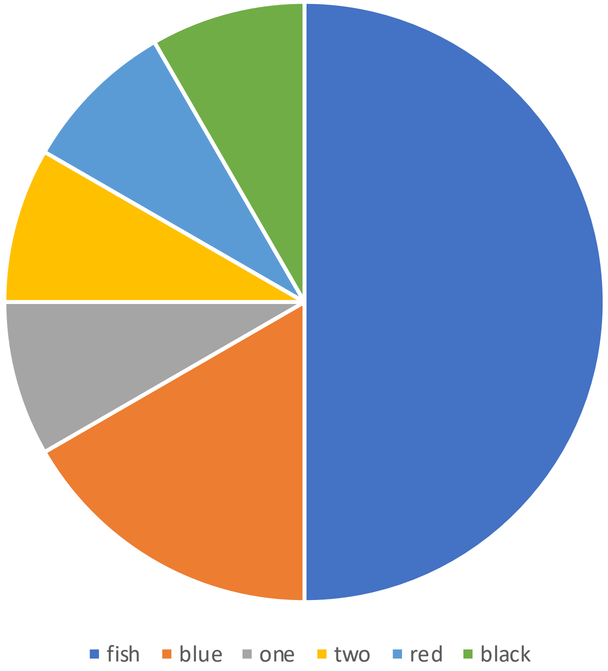
- Assuming a unigram model
  - $N = 12$
  - $|V| = 6$

Random Sampling

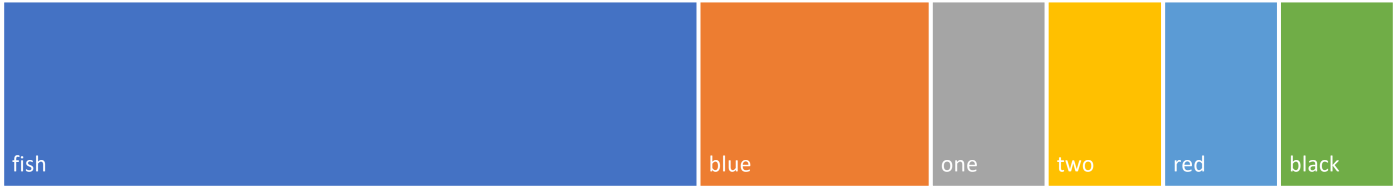
- Counts
  - ”fish” = 6
  - “blue” = 2
  - ”one” = 1
  - “two” = 1
  - ”red” = 1
  - “black” = 1

- Unigram Probs
  - ”fish” = 6/12
  - “blue” = 2/12
  - ”one” = 1/12
  - “two” = 1/12
  - ”red” = 1/12
  - “black” = 1/12

Random Sampling

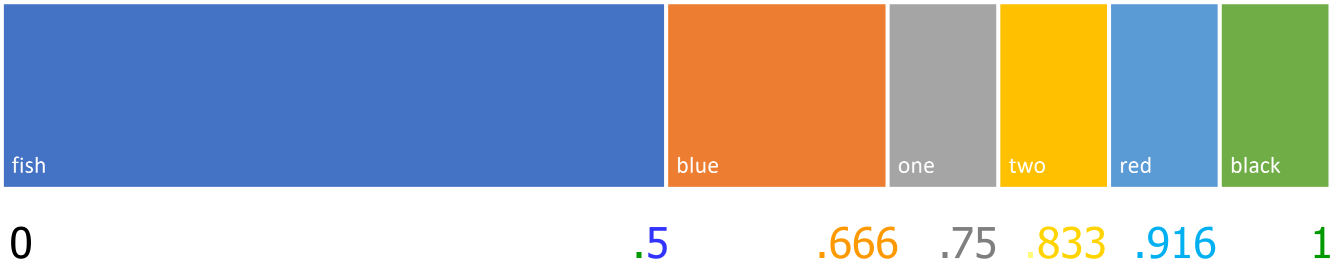


Random Sampling

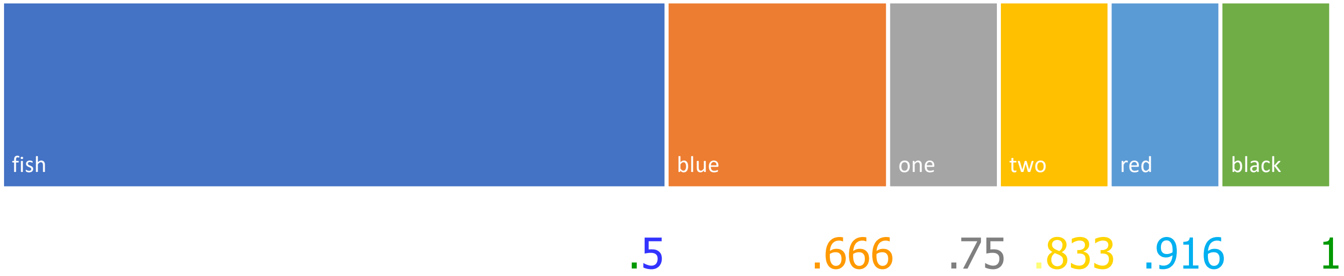


6/12 2/12 1/12 1/12 1/12 1/12

Random Sampling



Random Sampling



- Generate a random number between 0 and 1
- Find the bin that it falls in, return the word associated with that bin
- Repeat

- Numpy does this for you with random.choice()

The Chain Rule

$$\begin{aligned} P(w_{1:n}) &= P(w_1)P(w_2|w_1)P(w_3|w_{1:2})\ldots P(w_n|w_{1:n-1}) \\ &= \prod_{k=1}^{n} P(w_k|w_{1:k-1}) \end{aligned}$$

P(its water was so transparent)=

P(its)*

P(water|its)*

P(was|its water)*

P(so|its water was)*

P(transparent|its water was so)

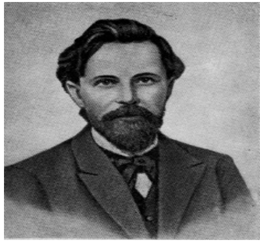
Markov Assumption

Replace each component in the product with an approximation (assuming a prefix of size N - 1)

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-N+1:n-1})$$

Bigram version

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-1})$$



The Chain Rule+Markov (Bigram)

$$P(w_1^n) \approx \prod_{k=1}^{n} P(w_k | w_{k-1})$$

P(its water was so transparent)=

P(its)*

P(water|its)*

P(was|water)*

P(so|was)*

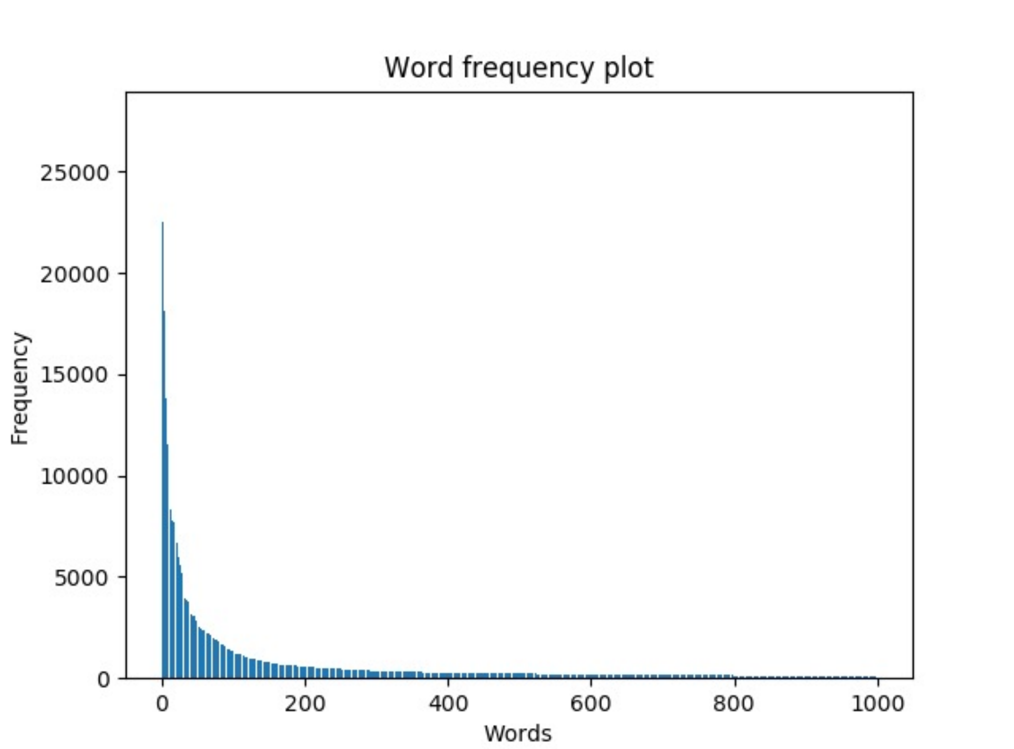
P(transparent|so)

Smoothing

- Back to Shakespeare
  - Shakespeare produced 300,000 bigram types out of V^2= 844 million possible bigrams… meaning 99.96% of the possible bigrams were never seen
  - Does that mean that any sentence that contains one of those bigrams should have a probability of 0?
  - For generation (Shannon game) it means we'll never emit those bigrams
  - For assigning probabilities to new sentences it means that if we run across a new bigram in the future then we have no choice but to assign it a probability of zero (and therefore to the sentence that contains it as well.

Zero Counts

- Some of those zeros are really
  - Things that really aren't ever goin
  - On the other hand, some of them
    been a little bigger they would have h
    - What would that count be in
- Zipf's Law (long tail phenomen
  - A small number of events occur w
  - A large number of events occur w
  - You can quickly collect statistics o
  - You might have to wait an arbitra
    events



- Result:
  - Our estimates are sparse! We have no counts at all for the vast number of things we want to estimate!

Notebook

Laplace Smoothing

- Also known as Add-One smoothing
- Just add one to all the counts!
- Very simple. For unigrams

  - MLE estimate: $P(w_i) = \frac{c_i}{N}$

  - Laplace estimate: $P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V}$

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

Laplace-Smoothed Bigram Counts

	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-Smoothed Bigram Probabilities

$$P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V}$$

	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

Reconstituted Bigram Counts

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Reconstituted Counts (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
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

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Big Change to the Counts!

- C(want to) went from 608 to 238!
- P(to|want) from .66 to .26!
- Discount d= c*/c
  - d for “chinese food” =.10!!! A 10x reduction
  - So, in general, Laplace is a very crude technique
- Despite this Laplace (add-1) is still used to smooth simple probabilistic models in NLP and IR, especially
  - For pilot studies
  - In document classification
  - In domains where the number of zeros isn’t so huge.

Types, Tokens and Fish

- Inspiration for many smoothing techniques comes from wildlife biology where 2 related problems arise
  1. Determining how many species occupy a particular area (how many types)
  2. Determining how many individuals of a given species are living in each area (tokens)

More Fish

- Imagine you are fishing
  - There are known to be 8 species of fish where you're fishing: carp, perch, whitefish, trout, salmon, eel, catfish, bass
    - Not sure where this fishing hole is...
- Up until now you have caught
  - 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel = 18 fish
- How likely is it that the next fish to be caught will be an eel?
- How likely is it that the next fish caught will be a member of one of the as yet to be seen species (bass or catfish)?
- Now how likely is it that the next fish caught will be an eel?

Slide adapted from Josh Goodman

Fishing Lesson

- We need to steal part of the observed probability mass to give it to the as yet unseen N-Grams. So the questions are:
    - How much to steal
    - How to redistribute it

Smoothing Concepts

- Backoff
  - Using lower order *N*-grams when counts are lacking for higher-order *N-grams*
- Interpolation
  - Mixing unigram, bigram, trigram probabilities
- Discounting
  - Stealing from the rich

Linear Interpolation

- Simple interpolation

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1 P(w_n|w_{n-2}w_{n-1}) \\ &+\lambda_2 P(w_n|w_{n-1}) \\ &+\lambda_3 P(w_n)\end{aligned} \qquad \sum_i \lambda_i = 1$$

- Lambdas conditional on context

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1(w_{n-2}^{n-1}) P(w_n|w_{n-2}w_{n-1}) \\ &+\lambda_2(w_{n-2}^{n-1}) P(w_n|w_{n-1}) \\ &+\lambda_3(w_{n-2}^{n-1}) P(w_n)\end{aligned}$$

How to set the Lambdas?

- Use a held-out corpus

Training Data	Held-Out Dev Data	Test Data

- Choose λs to maximize the probability of held-out data:
  - Fix the N-gram probabilities (using the training data)
  - Then search for λs that give largest probability to held-out set

Absolute Discounting

- Subtract a small fixed amount from all the observed counts (call that *d*)
- So instead of

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

- We'll use

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

Absolute Discounting

- That leaves two issues...
  - What's *d* ?
  - And how to redistribute it?

Absolute Discounting

- How much to subtract?
  - Split training data in ½
  - Bigrams that occurred twice in the first batch, occurred on average 1.25 times in the second batch.
  - Observe <u>how often on average</u> bigrams that occurred with count *M* in the first ½ occur in the other ½ of the training data.

Bigram count in training	Bigram count in heldout set
0	.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

Absolute Discounting w/ Interpolation

discounted bigram

Interpolation weight

$$P_{\text{AbsoluteDiscounting}}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} + \lambda(w_{i-1})P(w_i)$$

unigram prob

Caveats

- Smoothing N-gram counts is directed at N-grams consisting of words that have occurred.
- Dealing with unknown words (OOV) is a separate issue.
- The more data you have the less important the choice of smoothing method

Huge Models

- These days big organizations use massive corpora to build models.
- Models so large that smoothing becomes less relevant
  - Still an issue but you don’t need sophisticated methods.

Stupid Backoff

- Backoff without discounting.
  - Essentially don't worry about whether you have a proper probability distribution.

$$S(w_i|w_{i-k+1:i-1}) = \begin{cases} \dfrac{\text{count}(w_{i-k+1:i})}{\text{count}(w_{i-k+1:i-1})} & \text{if count}(w_{i-k+1:i}) > 0 \\ \lambda S(w_i|w_{i-k+2:i-1}) & \text{otherwise} \end{cases}$$

$$S(w) = \frac{count(w)}{N}$$