# Natural Language Processing

DSECL ZG565

Session 2: N-Gram Language Models

Dr. Ashish Kulkarni

BITS Pilani

April 17, 2021

# Session Content

1. Language Modeling
2. N-grams
3. Evaluating Language Models
4. Generalization and Zeros
5. Smoothing, Backoff and Interpolation

# Guess the next word!

Please turn your homework ...
What time does the sun ...
This is the house ...
Curiosity killed the ...

**Given some context, there are certain words that are more likely than others**

# This extends beyond words

leaning tower of Pisa *v.s.* tower Pisa of leaning

I saw some boys playing in the field *v.s.* playing I saw boys field the in some

**Certain combinations of words or sentences are more likely than others**

# Language Models

We formalize this intuition into models that:

- assign probability to each possible next word
- assign probability to a sequence of words or a sentence

These models are called language models or LMs

The sequence of words is called an n-gram

We will study how to use n-gram language models to estimate the probability of the last word of an n-gram given the previous words, and also to estimate probabilities of sentences.

# Why Language Models?

Speech recognition

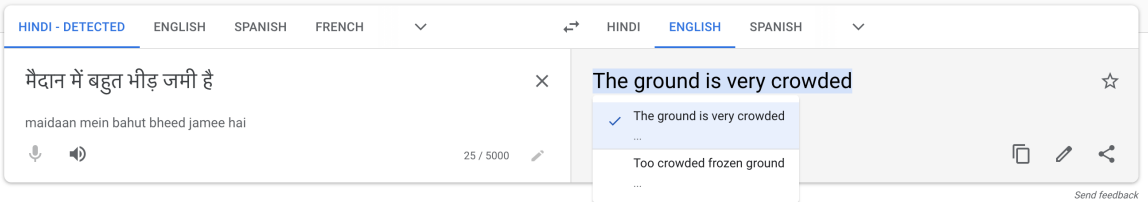
In India, eminence and success of lawyers go hand in hand.

india eminem's and success of lawyers go hand in hand

Spelling correction or Grammatical error correction

"Their are two exams"

Machine translation



# Probability of a sequence of words

| | |
|---|---|
| $P(X_i = \text{“}the\text{”})$ or $P(\text{“}the\text{”})$ | probability of a random variable taking a value “the” |
| $w_1 \dots w_n$ or $w_1^n$ | sequence of $n$ words |
| $P(X = w_1, \dots W = w_n) = P(w_1, \dots w_n)$ | Joint probability of a sequence of words |

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

Recall: Chain rule for probability

# How do we estimate $P(w_k|w_1^{k-1})$?

One way to do this is using **relative frequency counts**

$P(\text{the}|\text{its water is so transparent that}) = \frac{C(\text{its water is so transparent that the})}{C(\text{its water is so transparent that})}$
where, $C(\cdot)$ is the count operator

$$P(w_k|w_1^{k-1}) = \frac{P(w_1^k)}{P(w_1^{k-1})} \tag{2}$$

**Hard to get reliable estimates, especially, for large sequences!**

Check: `https://books.google.com/ngrams`

# N-gram language models

Instead of computing the probability of a word given its entire history, we can **approximate** the history by just the last few words - **Markovian assumption**

**Bigram model**: uses just the preceding word

$$P(w_k|w_1^{k-1}) \approx P(w_k|w_{k-1}) \tag{3}$$

**N-gram model**: uses the preceding $N-1$ words,

$$P(w_k|w_1^{k-1}) \approx P(w_k|w_{k-N+1}^{k-1}) \tag{4}$$

Fails to capture **long-distance dependencies**!
"The computer(s) which I had just put into the machine room on the fifth floor is (are) crashing"

# How do we estimate N-gram probabilities?

Maximum Likelihood estimation (MLE)!

Assuming we have access to a large corpus:

$$P(w_k|w_{k-1}) = \frac{C(w_{k-1}w_k)}{\sum_w C(w_{k-1}w)} = \frac{C(w_{k-1}w_k)}{C(w_{k-1})} \tag{5}$$

⟨s⟩ I am Sam ⟨/s⟩

⟨s⟩ Sam I am ⟨/s⟩

⟨s⟩ I do not like green eggs and ham ⟨/s⟩

$$P(I|\langle s\rangle) = \frac{2}{3} = 0.67 \quad P(Sam|\langle s\rangle) = \frac{1}{3} = .33 \quad P(am|I) = \frac{2}{3} = .67$$
$$P(\langle/s\rangle|Sam) = \frac{1}{2} = .5 \quad P(Sam|am) = \frac{1}{2} = .5 \quad P(do|I) = \frac{1}{3} = .33$$

Question: What will be the MLE N-gram parameter estimation?

# Another example

| | i | want | to | eat | chinese | food | lunch | spend |
|---|---|---|---|---|---|---|---|---|
| **i** | 0.002 | 0.33 | 0 | 0.0036 | 0 | 0 | 0 | 0.00079 |
| **want** | 0.0022 | 0 | 0.66 | 0.0011 | 0.0065 | 0.0065 | 0.0054 | 0.0011 |
| **to** | 0.00083 | 0 | 0.0017 | 0.28 | 0.00083 | 0 | 0.0025 | 0.087 |
| **eat** | 0 | 0 | 0.0027 | 0 | 0.021 | 0.0027 | 0.056 | 0 |
| **chinese** | 0.0063 | 0 | 0 | 0 | 0 | 0.52 | 0.0063 | 0 |
| **food** | 0.014 | 0 | 0.014 | 0 | 0.00092 | 0.0037 | 0 | 0 |
| **lunch** | 0.0059 | 0 | 0 | 0 | 0 | 0.0029 | 0 | 0 |
| **spend** | 0.0036 | 0 | 0.0036 | 0 | 0 | 0 | 0 | 0 |

Given:

| | |
|---|---|
| $P(i\|\langle s\rangle)$ | 0.25 |
| $P(\langle/s\rangle\|food)$ | 0.68 |

$P(\langle s\rangle$i want chinese food$\langle/s\rangle)$

$= P(i|\langle s\rangle)P(want|i)P(chinese|want)P(food|chinese)P(\langle/s\rangle|food)$

$= .25 \times .33 \times .0065 \times .52 \times .68$

$= .000019$

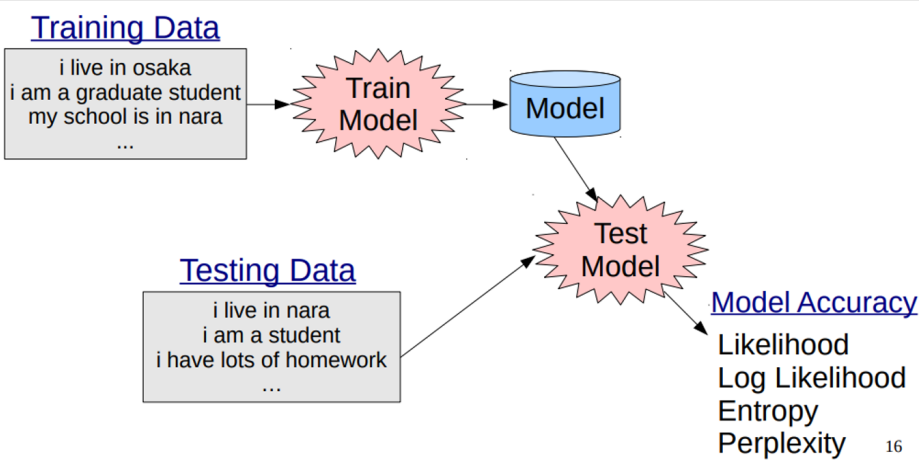
# Evaluation techniques

**Extrinsic evaluation**: embed the language model in an application and measure how much the application improves

**Intrinsic evaluation**: measure the quality of the language model independent of any application

- train the N-gram model parameters on a training set
- evaluate how well does this model "fit a test set"

# Experimental Setup



# Perplexity

The **perplexity** of a language model on a test set is the inverse probability of the test set, normalized by the number of words.
For a test set $W = w_1 w_2 \dots w_n$:

$$\begin{aligned} PP(W) &= P(w_1 w_2 \dots w_n)^{\frac{-1}{n}} \\ &= \sqrt[n]{\frac{1}{P(w_1 w_2 \dots w_n)}} \\ &= \sqrt[n]{\prod_{k=1}^{n} \frac{1}{P(w_k | w_1 \dots w_{k-1})}} \end{aligned} \tag{6}$$

- Lower the perplexity of a language model the better!
- the more information the n-gram gives us about the word sequence, the lower the perplexity
- An (intrinsic) improvement in perplexity does not guarantee an (extrinsic) improvement in the performance of a language processing task

# Language Model Generalization

N-grams do a progressively better job of modeling the training corpus as we increase the value of N

| | |
|---|---|
| 1 gram | –To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have<br>–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.<br>–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.<br>–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. A great banquet serv'd in;<br>–It cannot be but so. |

Figure: sentences randomly generated from four n-grams computed from Shakespeare's works

# Cross-Domain Generalization

| | |
|---|---|
| 1 gram | Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives |
| 2 gram | Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her |
| 3 gram | They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions |

Figure: sentences randomly generated from three N-gram models computed from 40 million words of the Wall Street Journal

Use a training corpus from a domain similar to that of the task being accomplished

# Sparsity

- any N-gram that occurred a sufficient number of times, we might have a good estimate of its probability
- any corpus being limited, some perfectly acceptable word sequences are bound to be missing, and therefore, get a zero probability

This is referred to as **sparsity** and this is a problem because:

- this indicates that we are **underestimating** the probability of all word sequences that might occur;
- if the probability of a word in the test set is 0, the entire probability of the test set is 0

# Out of Vocabulary Words

- Words that the model has not seen at the time of training are called **out-of-vocabulary (OOV) words**
- The percentage of OOV words that appear in the test set is called the **OOV rate**
- These are typically represented by a pseudo-word called $\langle \text{UNK} \rangle$

Fix the vocabulary in advance:

- convert in the training set any word not in this vocabulary to $\langle \text{UNK} \rangle$;
- estimate the probabilities for $\langle \text{UNK} \rangle$ from its count just like any other regular word in the training set

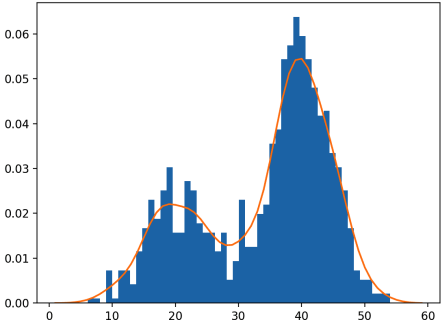
Create a vocabulary implicitly:

- replace by $\langle \text{UNK} \rangle$ all words that occur fewer than $n$ times in the training set; OR
- fix the vocabulary size $V$ in advance and choose the top $V$ words by frequency and replace the rest by $\langle \text{UNK} \rangle$

# What is Language Model Smoothing?

We are still left with words in our vocabulary that might appear in a test set in an unseen context.

**Smoothing** is a technique for assigning non-zero probabilities to such event by transferring to them some probability mass from more frequent events.



# Laplace Smoothing

Laplace smoothing merely adds 1 to each count (hence also called **add-one** smoothing)

$$P_{Laplace}(w_i) = \frac{c_i + 1}{N + V} \text{where, } c_i \text{ is the count of } w_i \text{ in the training corpus} \tag{7}$$

Alternatively, we can define an **adjusted count** $c^*$:

$$c_i* = (c_i + 1)\frac{N}{N + V} \tag{8}$$

We can now turn $c_i^*$ into a probability by normalizing by N.

The non-zero counts are **discounted** by a factor $d_c = c^*/c$ in order to get the probability mass that is assigned to the zero counts.

# Laplace Smoothing - Illustration

| | i | want | to | eat | chinese | food | lunch | spend |
|---|---|---|---|---|---|---|---|---|
| **i** | 0.002 | 0.33 | 0 | 0.0036 | 0 | 0 | 0 | 0.00079 |
| **want** | 0.0022 | 0 | 0.66 | 0.0011 | 0.0065 | 0.0065 | 0.0054 | 0.0011 |
| **to** | 0.00083 | 0 | 0.0017 | 0.28 | 0.00083 | 0 | 0.0025 | 0.087 |
| **eat** | 0 | 0 | 0.0027 | 0 | 0.021 | 0.0027 | 0.056 | 0 |
| **chinese** | 0.0063 | 0 | 0 | 0 | 0 | 0.52 | 0.0063 | 0 |
| **food** | 0.014 | 0 | 0.014 | 0 | 0.00092 | 0.0037 | 0 | 0 |
| **lunch** | 0.0059 | 0 | 0 | 0 | 0 | 0.0029 | 0 | 0 |
| **spend** | 0.0036 | 0 | 0.0036 | 0 | 0 | 0 | 0 | 0 |

Figure: Bigram probabilities for eight words from the BRP corpus

| | 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 |

Figure: Laplace smoothed bigram probabilities

| | 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 |

Figure: Laplace smoothed 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 |

Figure: Laplace smoothed adjusted counts

# Add-k Smoothing

Laplace smoothing causes a sharp change in counts and probabilities - **too much probability mass is transferred to the zeros!**

**Add-k smoothing**: Instead of adding 1 to each count, we add a fractional count $k$

$$P^*_{Add-k}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + k}{C(w_{n-1}) + kV} \tag{9}$$

$k$ is typically tuned on a dev or holdout set.

# Backoff and Interpolation

**Backoff**: "Backoff" to a lower-order N-gram if we have zero evidence for a higher-order N-gram

For instance, In case of a trigram language model, if we are estimating $P(w_n|w_{n-2}w_{n-1})$ but have no examples for $w_{n-2}w_{n-1}w_n$, we backoff to $P(w_n|w_{n-1})$ and to $P(w_n)$ if required.

**Interpolation**: Mix the probability estimates from all the N-gram estimators

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

$$\sum_i \lambda_i = 1$$

$\lambda$s are tuned on a holdout set

# Discounting in Backoff

In order for a backoff model to give a correct probability distribution, we have to **discount higher-order n-grams** to transfer some probability mass to the lower order n-grams.

This kind of backoff with discounting is called **Katz backoff**

$$P_{BO}(w_n|w_{n-N+1}^{n-1}) = \begin{cases} P^*(w_n|w_{n-N+1}^{n-1}) \text{ if } C(w_{n-N+1}^{n} > 0 \\ \alpha(w_{n-N+1}^{n-1})P_{BO}(w_n|w_{n-N+2}^{n-1}) \text{ otherwise.} \end{cases} \tag{10}$$

$\alpha$ is a function to distribute the probability mass to the lower order n-grams.

# Stupid Backoff

- Building large Web-scale language models is challenging (Recall: Google ngram!)
- Brants *et al.* (2007) show that with very large language models a much simpler algorithm may be sufficient. They proposed **stupid backoff**

**Stupid backoff**: gives up the idea of trying to make the language model a true probability distribution - no discounting!

$$S(w_i|w_{i-k+1}^{i-1}) = \begin{cases} \frac{count(w_{i-k+1}^{i})}{count(w_{i-k+1}^{i-1})} \text{ if } count(w_{i-k+1}^{i}) > 0 \\ \lambda S(w_i|w_{i-k+2}^{i-1}) \text{ otherwise} \end{cases} \tag{11}$$

# The End