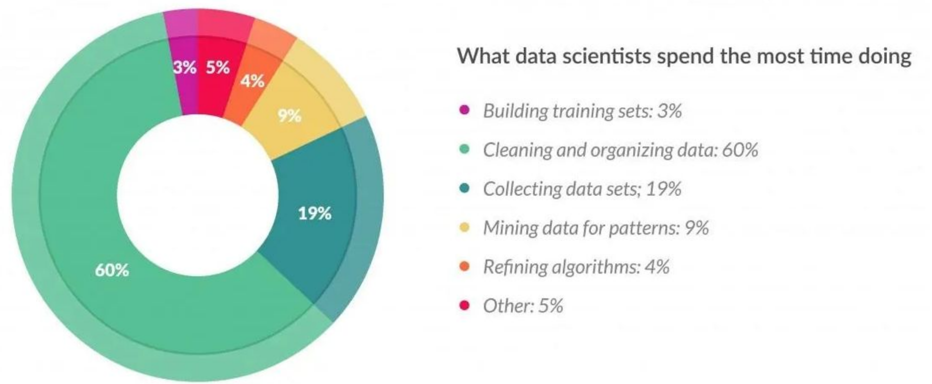
CS4248: Natural Language Processing

Lecture 3 — n-Gram Language Models

What do you want to learn?

- *"Understanding LLMs such ChatGPT"*
  - Provocative statement: Nobody really <u>understands</u> LLMs, i.e., <u>why/how</u> they work!
  - The way to understand LLMs requires a lot of background which we will cover
  - We end with an introduction into LLMs, but they are not and can not be the focus of CS4248
    (a dedicated graduate course covering LLMs is currently in the planning/preparation stage – stay tuned!)
  - In practice, fine-tuning LLMs is much more about proper data preparation than the actual training



Source: https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/

What do you want to learn?

- *HuggingFace, Langchain, Tensorflow, PyTorch, scikit-learn, numpy, etc.*
  - The lecture content focusing on the fundamental concept, not specific tools and libraries
  - We provide many practical examples in our series supplementary notebooks
  - You are free and encouraged to explore any available tools/frameworks/libraries for your project

Your Concerns — Our Comments

- *"I'm a total NLP/ML noob"*
  - CS4248 is a introduction / foundation course – we basically start from scratch
  - While some background knowledge is certainly useful, it's not a requirement
  - We only focus on nitty-gritty details required for this course (e.g., we do not cover backpropagation)

- *"I'm worried that there will be lots of math."*
  - Yes, there will be math, but nothing beyond fundamental concepts of algebra, probability, calculus
  - What need we need in this course, you will need in any computer/data science field!
  - We hope we cover the math bits in sufficient detail and clarity (if not, you can always ask!)

Your Concerns — Our Comments

- *"I've heard this course is hard!", "I'm afraid of the workload."*
  - Bias alert: We don't think that CS4248 is harder (or easier) than other courses
  - The assessment components are very similar to other course – participation marks are basically free marks :)
  - Consider assignments not just as an assessment component but as a learning experience

- *"I'm worried about the project."*
  - With reasonable effort, it is almost impossible to "fail" the project – we don't expect SOTA results :)
  - Basic suggestions: start early, continuous progress, regular team meetings (and/or with TA)
  - The project provides some flexibility to cater your background and interests
  - You can and should raise any inter-group conflicts incl. non-contributing members
    (there will be 2 rounds of peer review sessions using TEAMMATES!)

Recap of Week 02

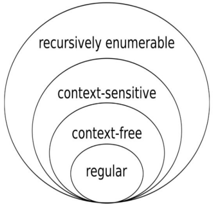
Relationship to Finite State Automata

- Equivalence
  - Regular Expressions describe **Regular Languages**
    (most restricted types of languages with respect to the Chomsky Hierarchy)
  - Regular Language = language accepted by a FSA

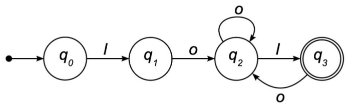
Regular Expression
l(o+l)+

Regular Language
{lol, loool, lolol, looolol, …}

Chomsky Hierarchy
(Source: Wikipedia)



Example: FSA that accepts the Regular Language described by the Regular Expression **l(o+l)+**



Minimum Edit Distance — Backtrace & Alignments

E	8	↙←↓ 9	↓ 8	↓ 7	↙←↓ 8	↓ 7	↓ 6	↙ 5
G	7	↙←↓ 8	↓ 7	↓ 6	↙←↓ 7	↓ 6	↙ 5	← 6
A	6	↙←↓ 7	↙↓ 6	↓ 5	↙←↓ 6	↙ 5	← 6	← 7
U	5	↙←↓ 6	↓ 5	↙ 4	← 5	← 6	←↓ 7	↙←↓ 8
G	4	↙←↓ 5	↓ 4	↙←↓ 5	↙←↓ 6	↙←↓ 7	↙ 6	← 7
N	3	↙←↓ 4	↓ 3	↙←↓ 4	↙←↓ 5	↙←↓ 6	↙←↓ 7	↙←↓ 8
A	2	↙←↓ 3	↙ 2	← 3	← 4	↙← 5	← 6	← 7
L	1	↙←↓ 2	↙←↓ 3	↙←↓ 4	↙←↓ 5	↙←↓ 6	↙←↓ 7	↙←↓ 8
#	0	1	2	3	4	5	6	7
	#	**S**	**A**	**U**	**S**	**A**	**G**	**E**

Quick quiz: Why do we choose the diagonal path here?

```
L A N G U * A G E
| | | | | | | | |
S A * * U S A G E
```

Complexity analysis
Time: O(n+m)

Tokenization — BPE Token Learner

corpus representation

6	n e w e s t _
5	l o w _
3	w i d e s t _
2	l o w e r _
1	l o n g e r _

vocabulary
d, e, g, i, l, n, o, r, s, t, w, _

merges

most frequent pair: **e** & **s** (9 occurrences)

corpus representation

6	n e w **es** t _
5	l o w _
3	w i d **es** t _
2	l o w e r _
1	l o n g e r _

vocabulary
d, e, g, i, l, n, o, r, s, t, w, _, **es**

merges
(e, s)

most frequent pair: **es** & **t** (9 occurrences)

Noisy Channel Model — Calculating/Estimating $P(x|w)$

$$
P(x|w) = \begin{cases} \dfrac{ins[w_{i-1}, x_i]}{count[w_i]} & \text{, if insertion} \\ \dfrac{del[w_{i-1}, w_i]}{count[w_{i-1}, w_i]} & \text{, if deletion} \\ \dfrac{sub[x_i, w_i]}{count[w_i]} & \text{, if substitution} \\ \dfrac{trans[w_i, w_{i+1}]}{count[w_i, w_{i+1}]} & \text{, if transposition} \end{cases}
$$

w_i = i-th character in the correct word w

x_i = i-th character in the misspelled word x

Outline

- **Language Models**
  - **Motivation**
  - Sentence Probabilities
  - Markov Assumption
  - Challenges
- Smoothing
  - Laplace Smoothing
  - Backoff & Interpolation
  - Kneser-Ney Smoothing
- Evaluating Language Models

Pre-Lecture Activity from Last Week

- Assigned Task
  - Post a 1–2 sentence answer to the following question into the L1 Discussion (you will find the thread on Canvas > Discussions)

“What do we mean when we talk about the probability of a sentence?”

Side notes:
- This task is meant as a warm-up to provide some context for the next lecture
- No worries if you get lost; we will talk about this in the next lecture
- You can just copy-&-paste others' answers, but his won't help you learn better

The probability that a sequence of words will form a coherent sentence with the correct context given prior knowledge.

How likely it is a sentence is grammatically correct.

Perhaps the probability that the sentence has a certain meaning?

LH

It means that the probability that the sentence is a valid or natural expression in a given language.

P(sentence) = 1 / # all possible sentences

It is the product of the probability of its words.

SH

The likelihood that a particular sequence of words forms a grammatically correct and meaningful statement within a given language.

It means how likely a sentence/phrase is expected to appear.

For example, if one of the training examples is 'I like cats', then 'cats' has a possibility to appear after 'I like' appears.

It refers to a sequence of words, and the probability that a word appears given the previous word in a sentence. Subsequently, all these successive probabilities can be calculated using chain rule to calculate the join probability of all word sequences to find the final sentence

The probability of a sentence is the probability that this sequence of words will appear given a random collection of words. For instance, the probability of the sentence "I am hungry" is P('I') * P('am' | 'I') * P('hungry' | 'I am').

Language Models — Motivation

- Which sentence makes more sense? S_1 or S_2?

Example 1:

S_1: *"on guys all I of noticed sidewalk three a sudden standing the"*

S_2: *"all of a sudden I noticed three guys standing on the sidewalk"*

Example 2:

S_1: *"the role was played by an ~~acress~~across famous for her comedic timing"*

S_2: *"the role was played by an ~~acress~~actress famous for her comedic timing"*

- But why?
  - Probability of S_2 higher than of S_1: $P(S_2) > P(S_1)$

➜ **Language Models** — Assigning probabilities to a sentence, phrase (or word)

Language Models — Basic Idea

- 2 basic notions of probabilities

(1) Probability of a sequence of words W

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

Example: $P(\text{“}remember\ to\ submit\ your\ assignment\text{”})$

(2) Probability of an upcoming word w_n

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

Example: $P(\text{“}assignment\text{”} \mid \text{“}remember\ to\ submit\ your\text{”})$

In this lecture: How to calculate these probabilities?

Language Models — Applications

- Language Models are fundamental for many NLP tasks
  - **Speech Recognition** $P(\text{“}we\ built\ this\ city\ on\ rock\ and\ roll\text{”}) > P(\text{“}we\ built\ this\ city\ on\ sausage\ rolls\text{”})$
  - **Spelling correction** $P(\text{“}\ldots\ has\ no\ mistakes\text{”}) > P(\text{“}\ldots\ has\ no\ \underline{mistaek}\text{”})$
  - **Grammar correction** $P(\text{“}\ldots\ has\ improved\text{”}) > P(\text{“}\ldots\ has\ \underline{improve}\text{”})$
  - **Machine Translation** $P(\text{“}I\ went\ home\text{”}) > P(\text{“}I\ went\ to\ home\text{”})$

Outline

- **Language Models**
  - Motivation
  - **Sentence Probabilities**
  - Markov Assumption
  - Challenges

- Smoothing
  - Laplace Smoothing
  - Backoff & Interpolation
  - Kneser-Ney Smoothing

- Evaluating Language Models

Probabilities of Sentences (more generally: sequence of words)

$$\left.\begin{array}{r} P(\text{“}remember\ to\ submit\ your\ assignment\text{”}) \\ P(\text{“}assignment\text{”}\,|\,\text{“}remember\ to\ submit\ your\text{”}) \end{array}\right\} \rightarrow$$ How to calculate those probabilities?

- Quick review: Chain Rule (allows the iterative calculation of joint probabilities)
  - Chain rule for 2 random events: $$P(A_1, A_2) = P(A_2|A_1) \cdot P(A_1)$$
  - Chain rule for 3 random events: $$\begin{aligned} P(A_1, A_2, A_3) &= P(A_3|A_1, A_2) \cdot P(A_1, A_2) \\ &= P(A_3|A_1, A_2) \cdot P(A_2|A_1) \cdot P(A_1) \end{aligned}$$
  - …

Probabilities of Sentences

- Chain rule — generalization to *N* random events

$$P(A_1, \ldots, A_N) = P(A_1) \cdot P(A_2|A_1) \cdot P(A_3|A_{1:\,2}) \cdot \cdots \cdot P(A_N|A_{1:\,N-1})$$
$$= \prod_{i=1}^{N} P(A_i|A_{1:\,i-1})$$

$i:j$ — sequence notations

➜ Chain rule applied to sequences of words

$$P(w_1, \ldots, w_N) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_{1:\,2}) \cdot \ldots \cdot P(w_N|w_{1:\,N-1})$$
$$= \prod_{i=1}^{N} P(w_i|w_{1:\,i-1})$$

🏃‍♀️🏃🏃‍♂️ Probably Correct? (5 mins)

Given two random variables X and Y with known probabilities $P(X)$ and $P(Y)$, compose as many statements with the tokens:

$$P(X) \quad P(Y) \quad P(Y|X) \quad P(X|Y) \quad > \quad < \quad =$$

And classify them as always correct, sometimes correct or never correct.

Post your answer to Canvas > Discussions > [In-Lecture Interaction] L1

(One student of your group can post the reply, and make sure to include your group members' names)

Probabilities of Sentences

- Calculating the probabilities using Maximum Likelihood Estimations

$$P(w_n|w_{1:n-1}) = \frac{Count(w_{1:n-1}w_n)}{\sum_w Count(w_{1:n-1}w)} = \frac{Count(w_{1:n})}{Count(w_{1:n-1})}$$

Quick quiz: Why does the denominator simplify like this?

Probabilities of Sentences — Example

(1) Application of Chain Rule

$$P(\text{“}remember\ to\ submit\ your\ assignment\text{”}) = \begin{aligned} & P(\text{“}remember\text{”}) \cdot \\ & P(\text{“}to\text{”} \mid \text{“}remember\text{”}) \cdot \\ & P(\text{“}submit\text{”} \mid \text{“}remember\ to\text{”}) \cdot \\ & P(\text{“}your\text{”} \mid \text{“}remember\ to\ submit\text{”}) \cdot \\ & P(\text{“}assignment\text{”} \mid \text{“}remember\ to\ submit\ your\text{”}) \end{aligned}$$

(2) Maximum Likelihood Estimation

$$P(\text{“}remember\text{”}) = \frac{Count(\text{“}remember\text{”})}{N}$$

$$P(\text{“}to\text{”} \mid \text{“}remember\text{”}) = \frac{Count(\text{“}remember\ to\text{”})}{Count(\text{“}remember\text{”})}$$

$$\ldots$$

$$P(\text{“}assignment\text{”} \mid \text{“}remember\ to\ submit\ your\text{”}) = \frac{Count(\text{“}remember\ to\ submit\ your\ assignment\text{”})}{Count(\text{“}remember\ to\ submit\ your\text{”})}$$

🤔 **Foreshadowing**:
Do you see any problems?

Probabilities of Sentences — Problems

$$P(\text{“}assignment\text{”} \mid \text{“}remember\ to\ submit\ your\text{”}) = \frac{Count(\text{“}remember\ to\ submit\ your\ assignment\text{”})}{Count(\text{“}remember\ to\ submit\ your\text{”})}$$

- Problem: (very) long sequences
  - Large number of entries in table with joint probabilities
  - A sequence (or subsequence) $w_{i:j}$ may not present in corpus

$\rightarrow$ $Count(w_{i:j}) = 0$ $\rightarrow$ $\prod_{n=1}^{N} P(w_n | w_{1:n-1}) = 0$

(we can ignore $\frac{0}{0}$ here; this can be handled in the implementation)

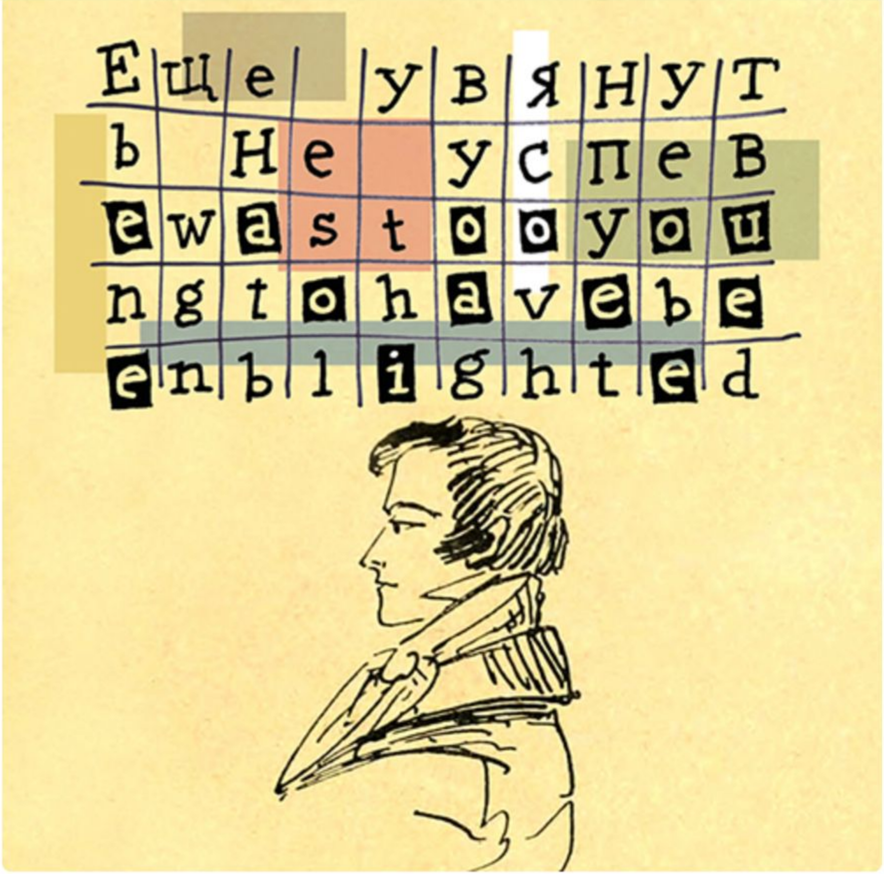
➜ Can we keep the sequences short?

Outline

- **Language Models**
  - Motivation
  - Sentence Probabilities
  - **Markov Assumption**
  - Challenges
- Smoothing
  - Laplace Smoothing
  - Backoff & Interpolation
  - Kneser-Ney Smoothing
- Evaluating Language Models

AMERICAN Scientist

“The first application of [A. A. Markov’s chains] was to a textual analysis of Alexander Pushkin's poem Eugene Onegin. Here a snippet of one verse appears (in Russian and English) along with Pushkin's own sketch of his protagonist Onegin.”



Markov Assumption

- Probabilities depend on only on the last k words

$$P(w_1, \ldots, w_N) = \prod_{n=1}^{N} P(w_n | w_{1:\, n-1}) = \prod_{n=1}^{N} P(w_n | w_{n-k:\, n-1})$$

- For our example:

$$P(\text{“assignment”} \mid \text{“remember to submit your”}) \approx P(\text{“assignment”} \mid \text{“your”})$$

$$P(\text{“assignment”} \mid \text{“submit your”})$$

$$P(\text{“assignment”} \mid \text{“to submit your”})$$

...

n-Gram Models (consider the only *n-1* last words)

Unigram (1-gram): $P(w_n|w_{1:\,n-1}) \approx\ ???$

Bigram (2-gram): $P(w_n|w_{1:\,n-1}) \approx\ ???$

Trigram (3-gram): $P(w_n|w_{1:\,n-1}) \approx\ ???$

n-Gram Models (consider the only *n-1* last words)

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

Bigram (2-gram): $P(w_n|w_{1:\,n-1}) \approx P(w_n|w_{n-1})$

Trigram (3-gram): $P(w_n|w_{1:\,n-1}) \approx P(w_n|w_{n-2}, w_{n-1})$

Quick quiz: How does this relate to context-sensitive or context-free?

n-Gram Models

Maximum Likelihood Estimation

Unigram (1-gram): $P(w_n|w_{1:\,n-1}) \approx P(w_n)$ $\qquad P(w_n) = \dfrac{Count(w_n)}{\#\ words}$

Bigram (2-gram): $P(w_n|w_{1:\,n-1}) \approx P(w_n|w_{n-1})$ $\qquad P(w_n|w_{n-1}) = \dfrac{Count(w_{n-1}w_n)}{Count(w_{n-1})}$

Trigram (3-gram): $P(w_n|w_{1:\,n-1}) \approx P(w_n|w_{n-2}, w_{n-1})$ $\qquad P(w_n|w_{n-1}, w_{n-2}) = \dfrac{Count(w_{n-2}w_{n-1}w_n)}{Count(w_{n-2}w_{n-1})}$

General MLE for *n*-grams: $P(w_i|w_{n-N+1:n-1}) = \dfrac{Count(w_{n-N+1:i})}{Count(w_{n-N+1:n-1})}$

- n-Gram models in practice
  - 3-gram, 4-gram, 5-gram models very common
  - The larger the n-grams, the more data required

🤔 **To Think About**:
How much more data?

n-Gram Models — Bigram Example

Example corpus with 3 sentences

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

$$P(\text{“}I\text{”} \mid \text{“}<s>\text{”}) = \frac{Count(\text{“}<s>\ \ I\text{”})}{Count(\text{“}<s>\text{”})} =$$

$$P(\text{“}am\text{”} \mid \text{“}I\text{”}) = \frac{Count(\text{“}I\ am\text{”})}{Count(\text{“}I\text{”})} =$$

$$P(\text{“}Sam\text{”} \mid \text{“}am\text{”}) = \frac{Count(\text{“}am\ Sam\text{”})}{Count(\text{“}am\text{”})} =$$

$$P(\text{“}</s>\text{”} \mid \text{“}Sam\text{”}) = \frac{Count(\text{“}Sam\ </s>\text{”})}{Count(\text{“}Sam\text{”})} =$$

n-Gram Models — Bigram Example

Example corpus with 3 sentences

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

$$P(``I"|``<s>") = \frac{Count(``<s> \ \ I")}{Count(``<s>")} = \frac{2}{3}$$

$$P(``am"|"I") = \frac{Count(``I \ am")}{Count(``I")} = \frac{2}{3}$$

$$P(``Sam"|"am") = \frac{Count(``am \ Sam")}{Count(``am")} = \frac{1}{2}$$

$$P(``</s>"|``Sam") = \frac{Count(``Sam </s>")}{Count(``Sam")} = \frac{1}{2}$$

n-Gram Models — Bigram Example (25,000 Movie Reviews)

$$P(\text{“}<s>\ \ i\ like\ the\ story\ \ </s>\text{”}) = ???$$

Unigram counts:

i	like	the	story
87,185	19,862	33,0867	11,094

Bigram counts:

	i	like	the	story
i	1	693	20	0
like	326	3	1,997	8
the	15	42	148	5171
story	23	16	16	0

n-Gram Models — Bigram Example (25,000 Movie Reviews)

$$P(\text{“}{<}s{>}\ \ i\ like\ the\ story\ \ {<}/s{>}\text{”}) = ???$$

Unigram counts:

i	like	the	story
87,185	19,862	33,0867	11,094

Bigram counts:

	i	like	the	story
i	0	693	20	0
like	326	0	1,997	8
the	15	42	0	5,171
story	23	16	16	0

Bigram probabilities:

	i	like	the	story
i	0.0	0.007949	0.000229	0.0
like	0.016413	0	0.100544	0.000403
the	0.000045	0.000127	0.0	0.015629
story	0.002073	0.001442	0.001442	0.0

Example calculation:

$$P(\text{“}like\text{”} \mid \text{“}i\text{”}) = \frac{Count(\text{“}i\ like\text{”})}{Count(\text{“}i\text{”})} = \frac{693}{87185} = 0.007949$$

n-Gram Models — Bigram Example (25,000 Movie Reviews)

Bigram probabilities:

	i	like	the	story
i	0.0	**0.007949**	0.000229	0.0
like	0.016413	0.0	**0.100544**	0.000403
the	0.000045	0.000127	0.0	**0.015629**
story	0.002073	0.001442	0.001442	0.0

Not in the table:

$$P(\text{``}i\text{''}\,|\,\text{``}{<}s{>}\text{''}) = 0.088198$$

$$P(\text{``}{<}/s{>}\text{''}\,|\,\text{``}story\text{''}) = 0.001262$$

$$P(\text{``}{<}s{>}\ i\ like\ the\ story\ {<}/s{>}\text{''}) = P(\text{``}i\text{''}\,|\,\text{``}{<}s{>}\text{''}) \cdot P(\text{``}like\text{''}\,|\,\text{``}i\text{''}) \cdot P(\text{``}the\text{''}\,|\,\text{``}like\text{''}) \cdot P(\text{``}story\text{''}\,|\,\text{``}the\text{''}) \cdot P(\text{``}{<}/s{>}\text{''}\,|\,\text{``}story\text{''})$$

$$P(\text{``}{<}s{>}\ i\ like\ the\ story\ {<}/s{>}\text{''}) = 0.088198 \cdot 0.007949 \cdot 0.100544 \cdot 0.015629 \cdot 0.001262$$

$$P(\text{``}{<}s{>}\ i\ like\ the\ story\ {<}/s{>}\text{''}) = 0.0000000139$$

Quick quiz: Why don't we need $P(\text{``}{<}s{>}\text{''})$?

n-Gram Models — Practical Consideration

- In general
  - Each $P(w_n|w_{1:n-1})$ rather small ➜ $\prod_{n=1}^{N} P(w_n|w_{1:n-1})$ very small
  - Risk of arithmetic underflow

➜ Always use an equivalent logarithmic format
  - Logarithm is a strictly monotonic function

$$P_1 \cdot P_2 \cdot P_3 \cdot \ldots P_N \propto \log\left(P_1 \cdot P_2 \cdot P_3 \cdot \ldots P_N\right)$$
$$= \log P_1 + \log P_2 + \log P_3 \cdot \ldots \log P_N$$

🏃‍♀️🏃🏃‍♂️ Quick Quiz (2 mins)

Given a **unigram** language model and the following two sentences S_1 and S_2

S_1: *"alice saw the accident"*

S_2: *"the accident alice saw"*

which sentence has the **higher probability**?

A $P(S_1) > P(S_2)$

B $P(S_1) < P(S_2)$

C $P(S_1) = P(S_2)$

D insufficient data

🏃‍♀️🏃🏃‍♂️ In-Lecture Activity (5 mins)

- Task: Calculate the Probability **P("saw"|"alice")** given the table of bigram counts below

- Post your answer to Canvas > Discussions > [In-Lecture] L1 … (Feb 2)
  (One student of your group can post the reply. Make sure to include your group members' names)

alice accident	5
saw alice	5
alice the	15
alice saw	20
saw the	25
accident saw	1
accident alice	2

Outline

- **Language Models**
  - Motivation
  - Sentence Probabilities
  - Markov Assumption
  - **Challenges**
- Smoothing
  - Laplace Smoothing
  - Backoff & Interpolation
  - Kneser-Ney Smoothing
- Evaluating Language Models

Handling OOV Words — Closed vs. Open Vocabulary

- Closed vocabulary
  - All strings contain words from a fixed vocabulary

  ➜ **No unknown words**

- Open Vocabulary
  - Strings may contain words that are not in the vocabulary (**OOV words**)
  - Examples: proper nouns, mismatching context

  ➜ **Counts might be 0** (even for individual words and not just for long(er) sequences of words)

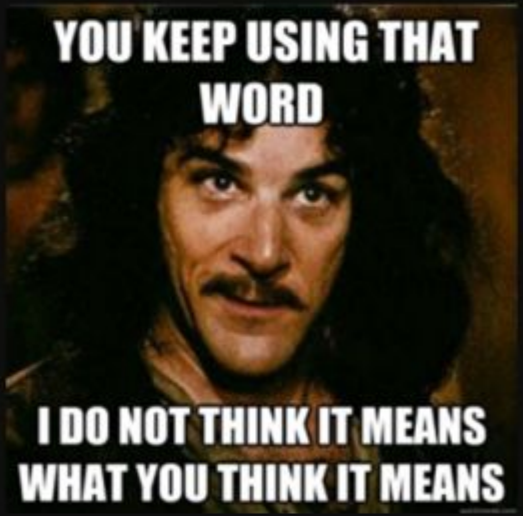
Movie review dataset — Unigram counts:

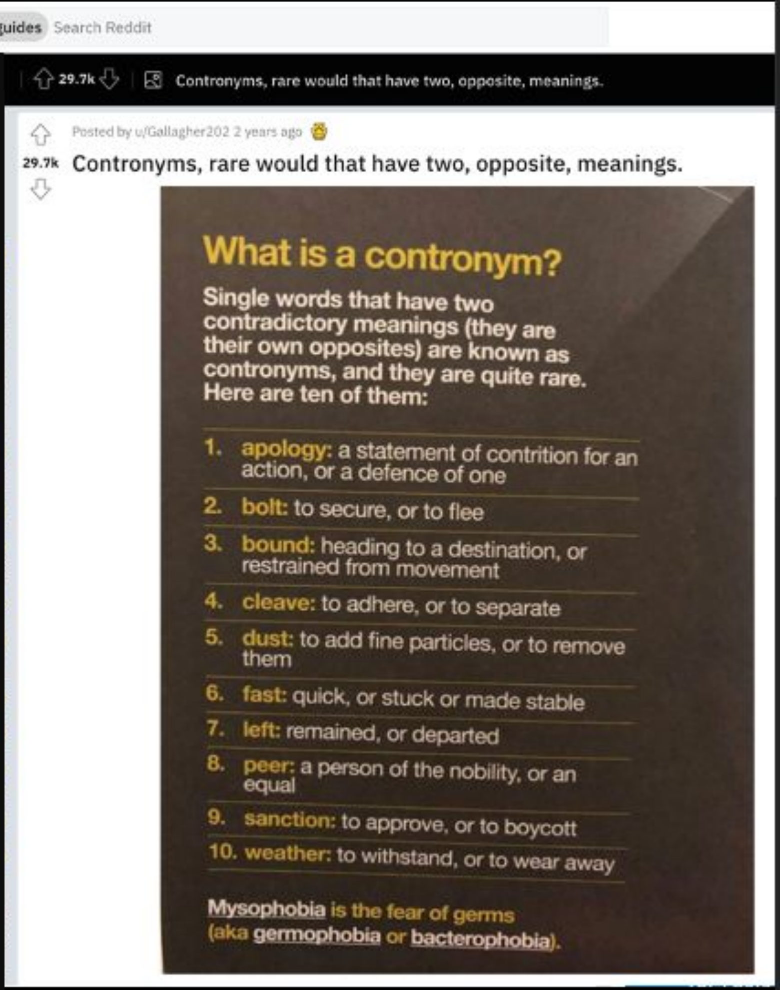
i	like	the	story	costner	einstein	planck	biden	integral	adverb	tensor	nlp
87,185	19,862	33,0867	11,094	67	20	**0**	**0**	27	**0**	**0**	**0**

Handling OOV Words — Alternatives

- Special token for OOV words
  - During normalization, replace all OOV words with a special token (e.g., `<UNK>`)
  - Estimate counts and probabilities for sequences involving `<UNK>` like for regular word
- Subword tokenization (e.g., with Byte-Pair Encoding (BPE) – Week 02)
  - Split texts into tokens smaller than words
  - Tokens are more likely to be frequent
- **Smoothing**

Break


YOU KEEP USING THAT WORD
I DO NOT THINK IT MEANS WHAT YOU THINK IT MEANS


reddit
r/coolguides
Search Reddit
29.7k
Contronyms, rare would that have two, opposite, meanings.
Posted by u/Gallagher202 2 years ago
29.7k
Contronyms, rare would that have two, opposite, meanings.
What is a contronym?
Single words that have two contradictory meanings (they are their own opposites) are known as contronyms, and they are quite rare. Here are ten of them:
1. apology: a statement of contrition for an action, or a defence of one
2. bolt: to secure, or to flee
3. bound: heading to a destination, or restrained from movement
4. cleave: to adhere, or to separate
5. dust: to add fine particles, or to remove them
6. fast: quick, or stuck or made stable
7. left: remained, or departed
8. peer: a person of the nobility, or an equal
9. sanction: to approve, or to boycott
10. weather: to withstand, or to wear away
Mysophobia is the fear of germs (aka germophobia or bacterophobia).


QI - Quite Interesting
April 11, 2018
A contronym is a single word with two definitions that are contradictory. For instance, dust can mean to cover with dust, but also to remove dust, and seed can mean to plant seeds, but also to remove seeds.
2.1K
220 Comments 624 Shares
Like
Comment
Share
Most relevant
Write a comment...
Angharad Jones
'Fast' can mean to move quickly or to be secured in place (stuck fast, fast asleep, fastened).
3 Replies
Josh Wi
"The alarm just went off"
"Well turn it off then"
John Pettigrew
Screen to show something or screen to hide something from view
Agnes Ambrosiani
In Swedish the word "dela" means both "share" and "divide". Leads to enormous confusion when talking about household finances...
1 Reply
Kevin Michael
Cleave can mean to both join and separate.
4 Replies
Dónal Maher
Does it include intonation?
Eg: "Will you come collect me?" "Sure"
That can be a very different answer if sarcastically intonated.....
1 Reply
Marley Bray
When someone says "I'm sure" it so often means they're not. "I'm sure I turned th the stove off". But it immediately denotes doubt.
Cass Mitchell
Victoria here's once for your language superstars. A bit more interesting than the usual synonyms and antonyms.
1 Reply

Outline

- Language Models
  - Motivation
  - Sentence Probabilities
  - Markov Assumption
  - Challenges
- **Smoothing**
  - **Laplace Smoothing**
  - Backoff & Interpolation
  - Kneser-Ney Smoothing
- Evaluating Language Models

Smoothing

- Basic idea
  - Avoid assigning probabilities of 0 to unseen n-grams
  - "Move" some probability mass from more frequent n-grams to unseen n-grams
  - Also called: **discounting**

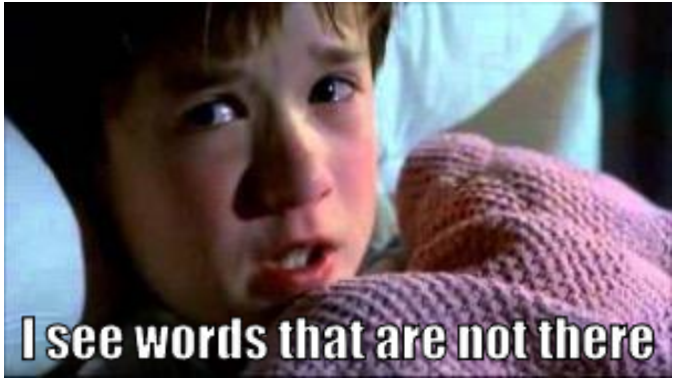


Photo Credits: Capture from *The Sixth Sense*, distributed by Buena Vista Pictures.

- Basic method: **Laplace Smoothing** (also: Add-1 Smoothing)
  - Example for bigrams

	i	like	the	story
i	0	693	20	0
like	326	0	1,997	8
the	15	42	0	5,171
story	23	16	16	0

Add 1

	i	like	the	story
i	1	694	21	1
like	327	1	1,998	9
the	16	43	1	5,172
story	24	17	17	1

Smoothing — Laplace Smoothing

- Calculating the probabilities

$$P_{Laplace}(w_n|w_{1:n-1}) = \frac{Count_{Laplace}(w_{1:n-1}w_n)}{\sum_w Count_{Laplace}(w_{1:n-1}w)}$$

$$= \frac{Count(w_{1:n-1}w_n) + 1}{\sum_w [Count(w_{1:n-1}w) + 1]}$$

$$= \frac{Count(w_{1:n-1}w_n) + 1}{Count(w_{1:n-1}) + V}$$

e.g., for bigrams: $P_{Laplace}(w_n|w_{n-1}) = \frac{Count(w_{n-1}w_n) + 1}{Count(w_{n-1}) + V}$

Smoothing — Laplace Smoothing

- Effects of smoothing on probabilities

Bigram probabilities (<u>without</u> Laplace Smoothing):

	i	like	the	story
i	0.0	0.007949	0.000229	0.0
like	0.016413	0	0.100544	0.000403
the	0.000045	0.000127	0.0	0.015629
story	0.002073	0.001442	0.001442	0.0

Bigram probabilities (<u>with</u> Laplace Smoothing):

	i	like	the	story
i	0.000006	0.004075	0.000123	0.000006
like	0.003175	0.000010	0.019401	0.000087
the	0.000039	0.000104	0.000002	0.012493
story	0.000255	0.000180	0.000180	0.000011

- Observations
  - No zero probabilities (duh!)
  - Some non-zero probabilities have changed quite a bit!

➜ For some n-grams: (arguably) too much probability gets moved to zero probabilities

Smoothing — Laplace Smoothing

- Effects of smoothing on counts
  - Question: What counts — without smoothing — would yield $P_{Laplace}(w_i|w_{i-1})$?

$$P_{Laplace}(w_n|w_{n-1}) = \frac{Count(w_{n-1}w_n)+1}{Count(w_{n-1})+V} = \frac{Count^*(w_{n-1}w_n)}{Count(w_{n-1})}$$

$$\rightarrow \; Count^*(w_{n-1}w_n) = (Count(w_{n-1}w_n)+1) \cdot \frac{Count(w_{n-1})}{Count(w_{n-1})+V}$$

Bigram counts (original):

	i	like	the	story
i	0	693	20	0
like	326	0	1,997	8
the	15	42	0	5,171
story	23	16	16	0

Bigram counts (adjusted):

	i	like	the	story
i	0.51	355.28	10.75	0.51
like	63.07	0.19	385.34	1.74
the	12.79	34.37	0.80	4133.5
story	2.83	2.00	2.00	0.12

Smoothing — Laplace Smoothing

- Laplace Discount
  - d_c — ratio of adjusted counts to the original counts
  - Only defined where original counts > 1

$$d_c = \frac{Count^*(w_{n-1}w_n)}{Count(w_{n-1}w_n)}$$

Laplace discounts:

	i	like	the	story
i		0.51	0.54	
like	0.19		0.19	0.22
the	0.85	0.82		0.80
story	0.12	0.13	0.13	

Add-*k* Smoothing

- Generalize Laplace (Add-1) Smoothing
  - Add k instead of 1
  - Set $0 < k \leq 1$

$$P_{add\text{-}k}(w_n|w_{n-1}) = \frac{Count(w_{n-1}w_n) + k}{Count(w_{n-1}) + kV}$$

Outline

- Language Models
  - Motivation
  - Sentence Probabilities
  - Markov Assumption
  - Challenges
- **Smoothing**
  - Laplace Smoothing
  - **Backoff & Interpolation**
  - Kneser-Ney Smoothing
- Evaluating Language Models

Backoff & Interpolation

- Intuition: Utilize less context if required
  - Assume we want to calculate $P(w_n|w_{n-2}, w_{n-1})$ but trigram $w_{n-2}w_{n-1}w_n$ is not in the dataset

(1) Backoff

- Make use if bigram probability $P(w_n|w_{n-1})$
- If still insufficient, use unigram probability $P(w_n)$

(2) Interpolation

- Estimate $P(w_n|w_{n-2}, w_{n-1})$ as a weighted mix of trigram, bigram, and unigram probabilities
- Learn weights λ_i from data
- In practice, better than Backoff

Linear Interpolation (example for trigrams)

- Simple interpolation

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

- λ_i conditioned on context

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

Backoff & Interpolation

- Learn weights λ_i from data — basic idea

(1) Collect held-out corpus
  - Additional corpus *or*
  - Split from initial corpus

(2) Calculate all n-gram probabilities
  - Calculation must not consider any held-out corpus!

(3) Find λ_i that maximizes $\hat{P}(w_n|w_{n-2}, w_{n-1})$ over held-out corpus
  - e.g., using Expectation-Maximization (EM) algorithm (not further discussed here)

Outline

- Language Models
  - Motivation
  - Sentence Probabilities
  - Markov Assumption
  - Challenges
- **Smoothing**
  - Laplace Smoothing
  - Backoff & Interpolation
  - **Kneser-Ney Smoothing**
- Evaluating Language Models

Kneser–Ney Smoothing

- Idea of Kneser–Ney Smoothing: **Absolute Discounting Interpolation**

Remove a fixed value from all bigram counts

Interpolation but with better estimates for unigram probabilities

$$P_{KN}(w_n|w_{n-1}) = \frac{max[Count(w_{n-1}w_n)-d,0]}{Count(w_{n-1})} + \lambda(w_{n-1})P_{KN}(w_n)$$

Note: We only look at a bigram language model in the following to keep the examples and notations easy. Kneser-Ney Smoothing is analogously defined for larger n-grams.

Kneser–Ney Smoothing — Absolute Discounting

- Absolute discounting
  - Remove fixed value d from bigram counts
    (typically: $0 < d < 1$)
  - Makes probability mass for unigrams available
  - Intuition

    If $Count(w_{n-1}w_n)$ is large, count is hardly affected

    If $Count(w_{n-1}w_n)$ is small, count is not that useful to begin with

just a fail-safe to avoid negative probabilities

$$\frac{max[Count(w_{n-1}w_n) - d, 0]}{Count(w_{n-1})}$$

➜ Question: How to pick the value(s) for d ?

Kneser–Ney Smoothing — Absolute Discounting

- Approach by Church and Gale (1991)
  - Compute bigram counts over large training corpus
  - Compute the counts of the same bigrams over a large test corpus
  - Compute the average count from the test corpus with respect to the count in the training corpus

Bigram count in training corpus	Bigram count in test corpus
0	0.000270
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

On average, a bigram that occurred 5 times in the training corpus occurred 4.21 times in the test corpus

➜ Set $d = 0.75$ (maybe a bit smaller for counts of 1 and 2)

Source: A comparison of the enhanced Good-Turing and deleted estimation methods for estimating probabilities of English bigrams (Church and Gale, 1991)

Kneser-Ney Smoothing — Interpolation with a Twist

- Motivation

$$P_{KN}(w_n|w_{n-1}) = \frac{max\left[Count(w_{n-1}w_n) - d, 0\right]}{Count(w_{n-1})} + \lambda(w_{n-1})P(w_n)$$

Using basic interpolation, that would just be the unigram probability

➜ **But is this actually a good idea?**

Predict the missing word:

"I can't see without my reading ______________"

$$\left\{ \begin{array}{l} \ldots \\ \text{“glasses”} \\ \ldots \\ \ldots \\ \text{“Kong”} \\ \ldots \end{array} \right.$$

If *"Hong Kong"* is very frequent:

$$P(\text{“}Kong\text{”}) > P(\text{“}glasses\text{”})$$

Kneser-Ney Smoothing — Interpolation with a Twist

- The difference between *"glasses"* and *"Kong"* — Intuition
  - *"glasses"* is preceded by many other words
  - *"Kong"* almost only preceded by *"Hong"*

➜ $P(w) =$ *"How likely is w ?"* … Maybe not most intuitive approach

- Alternative: $P_{KN}(w) =$ *"How likely is w to appear as a novel continuation?"*
  - $P_{KN}(w)$ is high ⇔ there are <u>many words</u> w' that form an existing bigram $w'w$
  - $P_{KN}(w)$ is low ⇔ there are <u>only few words</u> w' that form an existing bigram $w'w$

➜ How can we quantify this?

Kneser-Ney Smoothing — Interpolation with a Twist

- Calculating $P_{KN}(w)$

\# words w' that form an existing bigram $w'w$

$$P_{KN}(w) = \frac{|\{w' \; : \; Count(w'w) > 0\}|}{|\{(u,v) : Count(uv) > 0\}|}$$

total number of existing bigrams

normalization to ensure that $\sum_{n=1}^{N} P(w_n) = 1$

🏃‍♀️🏃🏃‍♂️ In-Lecture Activity (5 mins)

- Task: find 5+ words where you would expect that $P_{KN}(w) > P(w)$
  - Post your answer to Canvas > Discussions > [In-Lecture] L1 … (2 Feb)
    (one student of your group can post the reply, but include your group members' names)
  - We already used "Kong" as an example, so try to avoid "Francisco", "Angeles", "Aires", etc. :)
  - Optional: Think about how the context matters (e.g., travel blogs vs. movie reviews)

😎 **Pro Tip**: It's not a competition,
but about discussions and sharing ideas

Kneser-Ney Smoothing — Wrapping it Up

$$P_{KN}(w_n|w_{n-1}) = \frac{max\left[Count(w_{n-1}w_n) - d, 0\right]}{Count(w_{n-1})} + \underbrace{\lambda(w_{n-1})}_{\text{last missing puzzle piece}} P_{KN}(w_n)$$

- Normalizing factor λ
  - Required to account for the probability mass we have discounted

$$\lambda(w_{n-1}) = \underbrace{\frac{d}{Count(w_{n-1})}}_{\text{normalized discount}} \cdot \underbrace{\left|\{w' : Count(w_{n-1}w') > 0\}\right|}_{\text{\# words that can follow}}$$

normalized discount

\# words that can follow

= # words that have been discounted

= # times the normalized discount has been applied

Outline

- Language Models
  - Motivation
  - Sentence Probabilities
  - Markov Assumption
  - Challenges
- Smoothing
  - Laplace Smoothing
  - Backoff & Interpolation
  - Kneser-Ney Smoothing
- **Evaluating Language Models**

Evaluating Language Models

- A Language Model (LM) is considered good if
  - It assigns high probabilities to frequently occurring sentences
  - It assigns low probabilities to rarely occurring sentences

- 2 basic approaches to compare LMs

Extrinsic Evaluation

- Requires a downstream task
  (e.g., spell checker, speech recognition)
- Run downstream task with each LM and compare the results
- Can be very expensive & time-consuming

Intrinsic Evaluation

- Evaluate each LM on a test corpus
- Generally cheaper & faster
- Require intrinsic metric to compare LMs

➜ **Perplexity** (among other metrics)

Intrinsic Evaluation

- 3 core steps for an intrinsic evaluation

  (1) Train LM on a **training corpus**
  (i.e., compute the n-gram probabilities)

  (2) Tune parameters of LM using a **development corpus**
  (e.g., *k* in case of Add-*k* Smoothing)

  (3) Compute evaluation metric on **test corpus**
  (e.g., perplexity)

- Common corpus breakdown: 80/10/10 (80% training, 10% development, 10% test)

Perplexity — Intuition

- How easy is it to 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

- Unigrams are terrible at this game. Why?

Perplexity

- Perplexity — Definition
  - The best language model is the one that best predicts an unseen test set: highest $P(sentence)$
  - Inverse probability of test corpus W
  - Normalized by the number of words N in test corpus

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

$$= \sqrt[N]{\frac{1}{P(w_1, w_2, \ldots, w_N)}}$$

chain rule: $$= \sqrt[N]{\prod_{n=1}^{N} \frac{1}{P(w_n | w_1, \ldots, w_{n-1})}}$$

e.g., for bigrams: $$= \sqrt[N]{\prod_{n=1}^{N} \frac{1}{P(w_n | w_{n-1})}}$$

Minimizing perplexity ⇔ Maximizing probability

Perplexity — Intuition

- When is the perplexity **high** 😖?

Many n-grams are <u>frequent</u> in the training corpus but <u>rare</u> in the test corpus

↓

Very few high $P(w_n|w_{n-1})$ values over test corpus

Many n-grams are <u>rare</u> in the training corpus but <u>frequent</u> in the test corpus

↓

Many low $P(w_n|w_{n-1})$ values over test corpus

High perplexity $PP(W) = \sqrt[N]{\prod_{n=1}^{N} \frac{1}{P(w_n|w_{n-1})}}$

Perplexity — Practical Consideration

- In general
  - Each $P(w_n|w_{1:n-1})$ rather small ➜ $\prod_{n=1}^{N} P(w_n|w_{1:n-1})$ very small
  - Risk of arithmetic underflow

- Again, logarithm to the rescue

$$PP(W) = e^{\ln PP(W)}$$

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

$$= -\frac{1}{N} \ln \prod_{n=1}^{N} P(w_n|w_1, \ldots, w_{n-1})$$

$$= -\frac{1}{N} \sum_{n=1}^{N} \ln P(w_n|w_1, \ldots, w_{n-1})$$

e.g., for bigrams: $$= -\frac{1}{N} \sum_{n=1}^{N} \ln P(w_n|w_{n-1})$$

Perplexity — Toy Example

- Evaluation setup
  - Bigram LM trained over 25k movie reviews
  - Small test corpus W with $N = 12$

$$W = [$$
$$\text{"}\langle s\rangle\ i\ like\ good\ movies\ \langle /s\rangle\text{"},$$
$$\text{"}\langle s\rangle\ the\ story\ is\ funny\ \langle /s\rangle\text{"}$$
$$]$$

$$PP(W) = \sqrt[N]{\prod_{n=1}^{N} \frac{1}{P(w_n|w_{n-1})}} = \mathbf{40.1}$$

bigram	P(bigram)
"<s> i"	0.0882
"i like"	0.0079
"like good"	0.0013
"good movies"	0.0062
"movies </s>"	0.0034
"<s> the"	0.0990
"the story"	0.0156
"story is"	0.1138
"is funny"	0.0022
"funny </s>"	0.0081

Perplexity — Real-World Example

- Evaluation setup
  - Unigram, Bigram, Trigram LMs trained over *Wall Street Journal* articles
  - Training corpus: ~38 million words (~20k unique words)
  - Test corpus: ~1.5 million words

	Unigram	Bigram	Trigram
Perplexity	962	170	109

Source: https://web.stanford.edu/~jurafsky/slp3/3.pdf

🏃‍♀️🏃🏃‍♂️ Quick Quiz (2 mins)

What are the (**minimum**, **maximum**) possible values for perplexity?

A (0, ∞)

B (1, ∞)

C (0, V)

D (1, V)

v = size of vocabulary

Summary

- Language Models — assigning probabilities to sentences
  - Very important concept for many NLP tasks
  - Different methods to compute sentence probabilities
    (here: n-grams; later we come back to them using neural networks)

- n-gram Language Models
  - Intuitive training ➜ Maximum Likelihood Estimations
  - Main consideration: zero probabilities due to
    large n-grams and/or open vocabularies

Markov Assumption to limited
size of considered n-grams

Focus here: **Smoothing**
(maybe with backoff & interpolation)

In practice, typically a combination
of these and similar approaches

Outlook for Next Week: Text Classification

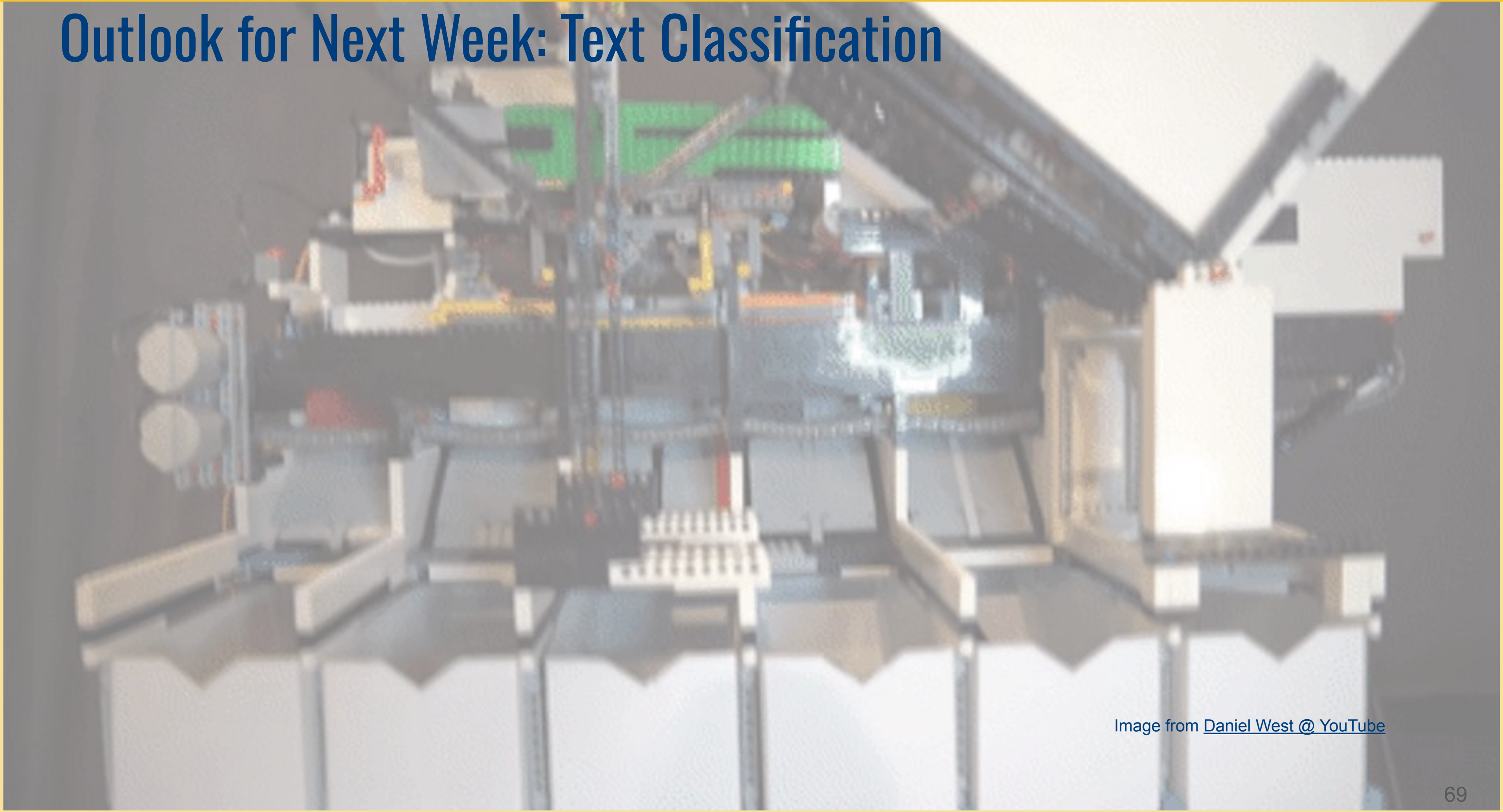


Image from [Daniel West @ YouTube]

Pre-Lecture Activity for Next Week

- Assigned Task (due before Feb 9)
  - Post a 1–2 sentence answer to the following question in the Pre-Lecture forum.
    (you will find the thread on Canvas > Discussions > [Pre-Lecture])

"When we want to evaluate classifiers,
*why is **accuracy** alone often not a good metric?"*

Side notes:

- This task is meant as a warm-up to provide some context for the next lecture
- No worries if you get lost; we will talk about this in the next lecture
- You can just copy-&-paste others' answers but this won't help you learn better