Text Mining and Analytics

Session 3: **N-Gram and Language Models**

Instructor: Behrooz Mansouri
Spring 2024, University of Southern Maine

Which one is more Likely?

Consider probability for the following three strings

- S_1= Portland Museum of Art
- S_2= Art of Portland Museum
- S_3= of Museum Art Portland

Which sentence has the highest probability of being seen in a document?

Language Model

A model for how humans generate language

- Places a probability distribution over any sequence of words
    - How likely is a given string (observation) in a given “language”
- By construction, it also provides a model for generating text according to its distribution

Used in many language-oriented tasks, e.g.,

● Machine translation:	P(high winds tonight)	>	P(large winds tonight)
● Spelling correction:	P(about 15 minutes)	>	P(about 15 minuets)
● Speech recognition:	P(I saw a van)	>>	P(eyes awe of an)

Language Modeling Problem

A language model over a vocabulary V assigns probabilities to strings drawn from V*

- Finite vocabulary (e.g., words or characters)
- Infinite set of sequences

Probabilistic Language Modeling

Goal: compute the probability of a sentence or sequence of words:

$$P(W) = P(w_1, w_2, w_3, w_4, w_5 \ldots w_n)$$

Related task: probability of an upcoming word:

$$P(w_5 | w_1, w_2, w_3, w_4)$$

A model that computes either of these:

$P(W)$ or $P(w_n | w_1, w_2 \ldots w_{n-1})$ is called a language model

Computing P(W)

How to compute this joint probability:

P(sequence of words)

Intuition: let's rely on the Chain Rule of Probability

Recall the definition of conditional probabilities

$P(B|A) = P(A,B)/P(A)$ Rewriting: $P(A,B) = P(A)P(B|A)$

The Chain Rule

More variables: P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)

The Chain Rule in General

$$P(x_1,x_2,x_3,\ldots,x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1,x_2)\ldots P(x_n|x_1,\ldots,x_{n-1})$$

For words in sentence $P(w_1 w_2 \ldots w_n) = \prod_i P(w_i \mid w_1 w_2 \ldots w_{i-1})$

e.g.,

P(“Joe knows Huffman”) = P(Joe) × P(knows|Joe) × P(Huffman|Joe knows)

Estimating the Probabilities: Counting Words

Probabilities are based on counting things (in text mining things=tokens)

Could we just count and divide?

$$P(\text{Coding} \mid \text{Joe knows Huffman}) = \frac{\text{Count (Joe knows Huffman Coding)}}{\text{Count (Joe knows Huffman)}}$$

Estimating the Probabilities: Counting Words

Probabilities are based on counting things (in text mining things=tokens)

Could we just count and divide?

$$P(\text{Coding} \mid \text{Joe knows Huffman}) = \frac{\text{Count (Joe knows Huffman Coding)}}{\text{Count (Joe knows Huffman)}}$$

- No! Too many possible sentences!
- We'll never see enough data for estimating these

Markov Assumption

Markov Assumption

The probability of a word in a sequence depends only on a fixed number of preceding words, rather than the entire history of the sequence

P(Coding | Joe knows Huffman) ≈ P(Coding | Huffman)

P(Coding | Joe knows Huffman) ≈ P(Coding | knows Huffman)

Andrey Markov

In other words, we approximate each component in the product

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i \mid w_{i-k} \ldots w_{i-1})$$

$$P(w_i \mid w_1 w_2 \cdots w_{i-1}) \approx P(w_i \mid w_{i-k} \cdots w_{i-1})$$

Unigram Language Model

How do we build probabilities over sequences of terms?

$$P(t_1t_2t_3t_4)=P(t_1)P(t_2|t_1)P(t_3|t_1t_2)P(t_4|t_1t_2t_3)$$

Unigram LM

- A unigram language model throws away all conditioning context, and estimates each term independently
- Generate a piece of text by generating each word independently

$$P_{uni}(t_1t_2t_3t_4)=P(t_1)P(t_2)P(t_3)P(t_4)$$

Example

$S_1 = P_{uni}(\text{“text mining course is great”}\ |M)$

$S_2 = P_{uni}(\text{“text mining course is boring”}|M)$

Model M

0.4 text

0.3 mining

0.5 course

0.2 is

0.8 great

0.05 boring

Example

$S_1 = P_{uni}(\text{“text mining course is great”}\ |M)$

$S_2 = P_{uni}(\text{“text mining course is boring”}|M)$

<u>Model M</u>

0.4 text

0.3 mining

0.5 course

0.2 is

0.8 great

0.05 boring

$P_{uni}(\text{“text mining course is great”}|M)$

$= 0.4 \times 0.3 \times 0.5 \times 0.2 \times 0.8 = 0.0096$

$P_{uni}(\text{“text mining course is boring”}|M)$

$= 0.4 \times 0.3 \times 0.5 \times 0.2 \times 0.05 = 0.0006$

$P_{uni}(S_1|M) > P_{uni}(S_2|M)$

Example

$S_1 = P_{uni}(\text{"text mining course is great"}\ |M)$

$S_2 = P_{uni}(\text{"text mining course is boring"}|M)$

<u>Model M</u>

0.4 text

0.3 mining

0.5 course

0.2 is

0.8 great

0.05 boring

<u>Model M2</u>

0.4 text

0.3 mining

0.5 course

0.2 is

0.1 great

0.9 boring

$P_{uni}(S_1|M_1) > P_{uni}(S_2|M_1)$

$P_{uni}(S_2|M_2) > P_{uni}(S_1|M_2)$

Bigram Language Model (2-gram)

Condition on the previous word: $P(w_i \mid w_1 w_2 \cdots w_{i-1}) \approx P(w_i \mid w_{i-1})$

$S_1 = P_{bi}$(“text mining course is great” |M)

$S_2 = P_{bi}$(“text mining course is boring”|M)

<u>Model M</u>

0.4 text|<s>

0.3 mining|text

0.5 course|mining

0.2 is|course

0.8 great|is

0.05 boring|is

P_{uni}(“text mining course is great”|M)

$= 0.4 \times 0.3 \times 0.5 \times 0.2 \times 0.8 = 0.0096$

P_{uni}(“text mining course is boring”|M)

$= 0.4 \times 0.3 \times 0.5 \times 0.2 \times 0.05 = 0.0006$

$P_{uni}(S_1|M) > P_{uni}(S_2|M)$

N-gram Models

We can extend to trigrams, 4-grams, 5-grams

In general, this is an insufficient model of language because language has long-distance dependencies:

- The computer which I had just put into the machine room on the fifth floor crashed

But we can often get away with N-gram models for specific tasks

Estimating N-grams

The Maximum Likelihood Estimate

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

Maximum Likelihood Estimation (MLE): estimating the parameters of an assumed probability distribution, given some observed data

[The Epic Story of Maximum Likelihood]()

Example (1): Berkeley Restaurant Project (BeRP)

A dialogue system from the last century that answered questions about a database of restaurants in Berkeley, California

Here are some text-normalized sample user queries

- mid priced thai food is what i’m looking for
- can you give me a listing of the kinds of food that are available
- i’m looking for a good place to eat breakfast
- when is caffe venezia open during the day

Example (1): Berkeley Restaurant Project (BeRP)

Bigram counts for eight of the words (V = 1446) in the Berkeley Restaurant Project corpus of 9332 sentences. Zero counts are in gray

Numbers: Row followed by Column

	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

Example (1): Berkeley Restaurant Project (BeRP)

Set of unigram counts

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

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

The bigram probabilities after normalization (dividing each cell by the appropriate unigram for its row)

	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

Example (1): Berkeley Restaurant Project (BeRP)

Using Maximum Likelihood Estimate (MLE)

P(<s> I want english food </s>) =
P(I|<s>)
× P(want|I)
× P(english|want)
× P(food|english)
× P(</s>|food)
= .000031

	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

Example (2)

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

<s> I am Sam </s>

<s> Sam I am </s>

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

P(I|<s>) = ?
P(Sam|<s>) = ?
P(am|I) = ?

P(</s>|Sam) = ?
P(Sam|am) = ?
P(do|I) = ?

Example (2)

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

<s> I am Sam </s>

<s> Sam I am </s>

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

$P(I|\text{<s>}) = \frac{2}{3} = 0.67$

$P(Sam|\text{<s>}) = \frac{1}{3} = 0.33$

$P(am|I) = \frac{2}{3} = 0.67$

$P(\text{</s>}|Sam) = \frac{1}{2} = 0.5$

$P(Sam|am) = \frac{1}{2} = 0.5$

$P(do|I) = \frac{1}{3} = 0.33$

How Good is Language Model?

Does our language model prefer good sentences to bad ones?

- Assign higher probability to “real” or “frequently observed” sentences than “ungrammatical” or “rarely observed” sentences

We train parameters of our model on a training set

We test the model’s performance on data we have not seen

- A test set is an unseen dataset that is different from our training set, totally unused
- An evaluation metric tells us how well our model does on the test set

Extrinsic Evaluation of N-gram Models

The best way to evaluate the performance of a language model is to embed it in an application and measure how much the application improves. Such end-to-end evaluation is called extrinsic evaluation

- Put each model in a task
  - spelling corrector, speech recognizer, MT system
- Run the task, get an accuracy for A and for B
  - How many misspelled words corrected properly
  - How many words translated correctly
- Compare accuracy for A and B

Practical Issues

We always represent and compute language model probabilities in log format as log probabilities

- Avoid underflow
- (also adding is faster than multiplying)

$$\log(p_1 \times p_2 \times p_3 \times p_4) = \log p_1 + \log p_2 + \log p_3 + \log p_4$$

Generalization and Zeros

Problem with Zero Probability

Training set:

... denied the allegations

... denied the reports

... denied the claims

... denied the request

Test set

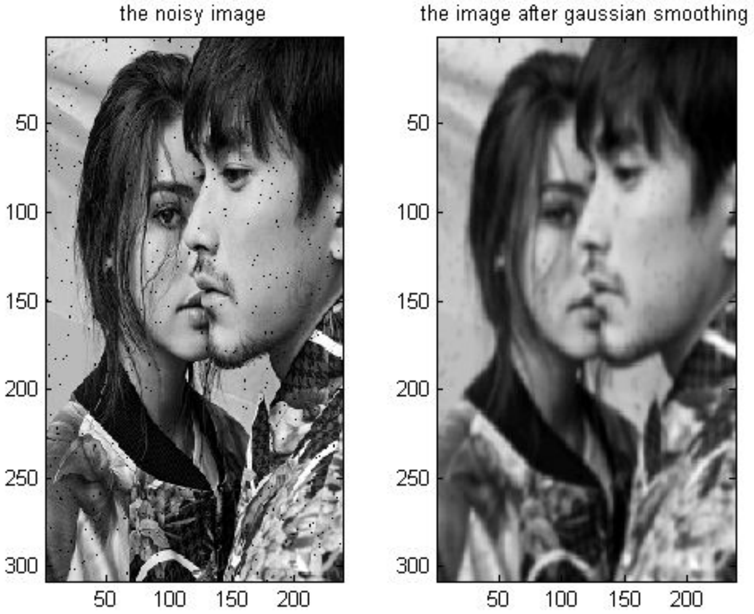
... denied the offer

... denied the loan

P(“offer” | denied the) = 0

Bigrams with zero probability: will assign 0 probability to the test set

Smoothing for Images



Idea of Smoothing

We can view a document as words sampled from the author's mind

- High-frequency words (e.g., rocky, apollo, boxing) are important
- Low-frequency words (e.g., shot, befriended, checks) are arbitrary

The author chose these, but could have easily chosen others

So, we want to allocate some probability to unobserved indexed-terms and discount some probability from those that appear in the document

Laplace Smoothing (Add-One Estimation)

Pretend we saw each word one more time than we did (i.e. just add one to all the counts)

MLE estimate:

$$P_{MLE}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

Add-1 estimate:

$$P_{Add-1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

Simple, but does not work well in practice

Backoff and Interpolation

Sometimes it helps to use less context (generalization)

- Condition on less context for contexts you have not learned much about

Backoff:

- Use trigram if you have good evidence (no Zero evidence)
- Otherwise bigram, otherwise unigram

Interpolation: mix unigram, bigram, trigram

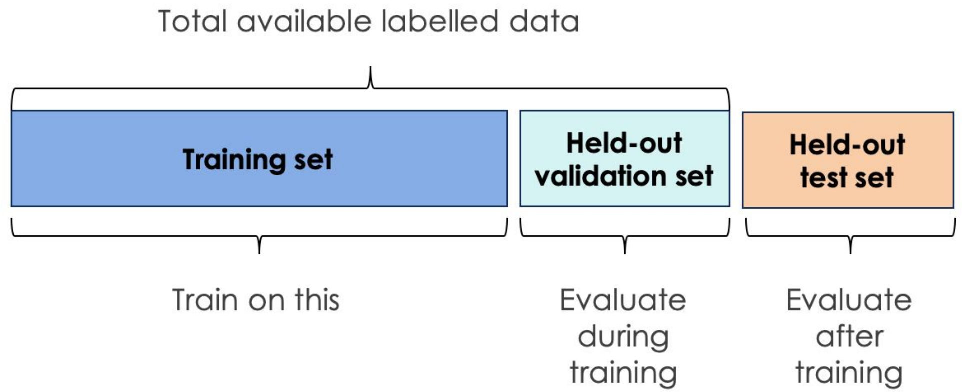
Linear interpolation

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

$$\sum_i \lambda_i = 1$$

Interpolation works better

Setting the Lambdas



Use a held-out corpus

Choose λs to maximize the probability of held-out data:

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

$$\log P(w_1 \ldots w_n \mid M(\lambda_1 \ldots \lambda_k)) = \sum_i \log P_{M(\lambda_1 \ldots \lambda_k)}(w_i \mid w_{i-1})$$

Out of Vocabulary

Define an unknown word token <UNK>

Training of <UNK> probabilities

- Create a fixed lexicon L of size V
- Any training word not in L changed to <UNK>
- Train language model probabilities as if <UNK> were a normal word

At decoding time

- Use <UNK> probabilities for any word not in training

Next Session

Vector Semantics

We will explore

- Lexical Semantics
- Vector Semantics
- Words and Vectors
- Cosine Similarity
- TF-IDF and Pointwise Mutual Information

To do:

- Assignment 1
- Reading: Chapter 3 of Jurafsky book
- Prepare for Quiz 1