innovate achieve lead

Natural Language Processing

Dr. Chetana Gavankar, Ph.D,
IIT Bombay-Monash University Australia
Chetana.gavankar@pilani.bits-pilani.ac.in

BITS Pilani
Pilani Campus

innovate achieve lead

BITS Pilani

Pilani Campus

Session 4

These slides are prepared by the instructor, with grateful acknowledgement of Jurafsky and Martin and many others who made their course materials freely available online.

Session Content

- Language Models
- Real world applications of language modelling
- How to build language models
- Evaluation of Language models
- Perplexity
- Smoothing, Interpolation and Back off

Language modelling

A model that computes either of these:

Probability of a sentence (P(W)) or

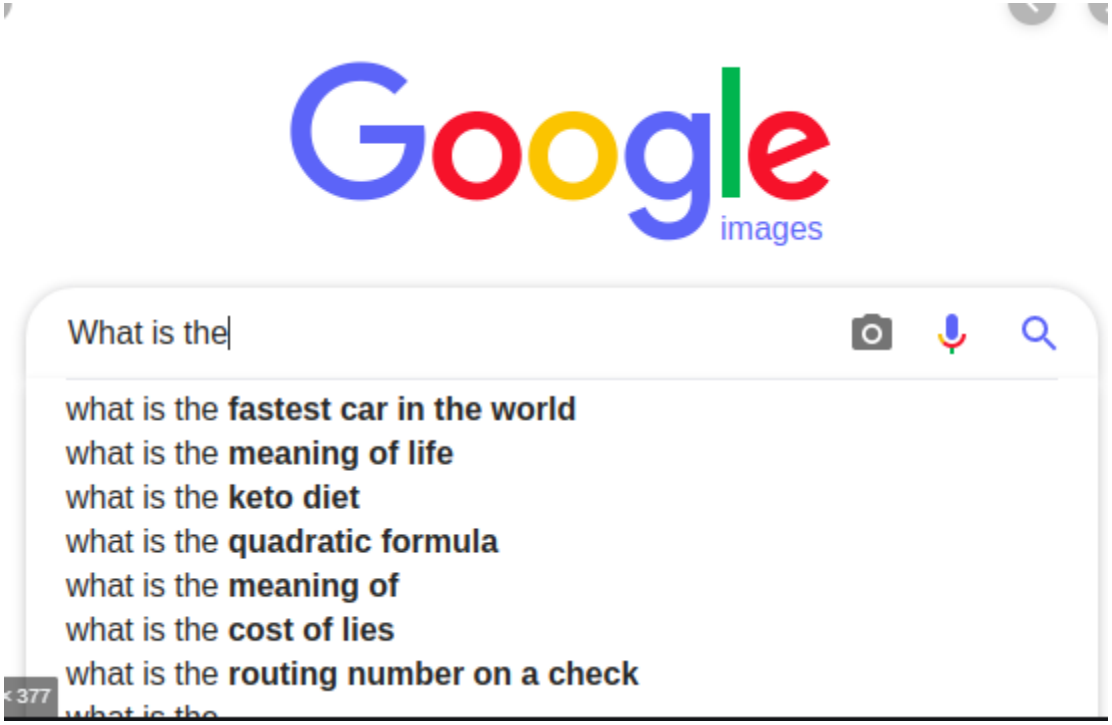
Probability of an upcoming word ($P(w_n|w_1,w_2...w_{n-1})$)

is called a **language model**.

Simply we can say that

A language model learns to predict the probability of a sequence of words.

Example


Google
images
What is the
what is the fastest car in the world
what is the meaning of life
what is the keto diet
what is the quadratic formula
what is the meaning of
what is the cost of lies
what is the routing number on a check

Language Models

Machine Translation

1.Machine Translation:

Machine translation system is used to translate one language to another language

P(**high** winds tonight) > P(**large** winds tonight)

2.Spell correction

P(about fifteen **minutes** from) > P(about fifteen **minuets** from)

3.Speech Recognition

Speech recognition is the ability of a machine or program to identify words and phrases in spoken language and convert them to a machine-readable format.

Example:

P(I saw a van) >> P(eyes awe of an)

How to build a language model

- Recall the definition of conditional probabilities

  p(B|A) = P(A,B)/P(A) Rewriting: **P(A,B) = P(A)P(B|A)**

- 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

$$P(x_1,x_2,x_3,\ldots,x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1,x_2)\ldots P(x_n|x_{n-1},x_{n-2}\ldots\ldots,x_1)$$

The Chain Rule applied to compute joint probability of words in sentence

$$P(w_1\, w_2 \ldots \ldots w_n) = \prod_{k=1}^{n} P(w_k \mid w_1^{k-1})$$

For ex:

P("its water is so transparent") =

P(its) × P(water|its) × P(is|its water)

× P(so|its water is) × P(transparent|its water is so)

Where:

P(its)= #(its)

P(water | its) = #(its water) / #(its)

P(is | its water) =#(its water is) / #(its water)

P(so | its water is) =#(its water is so) / #(its water is)

P(the | its water is so transparent) = #(its water is so transparent) / #(its water is so)

Markov Assumption

- Simplifying assumption:

limit history of fixed number of words, n

Andrei Markov

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{that})$$

or

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{transparent that})$$

Markov Assumption

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

In other words, we approximate each component in the product

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

N-gram Language models

- N gram is a sequence of tokens(words)
- Unigram language model(N=1)

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i)$$

Example:

P(I want to eat healthy food)≈ P(I)P(want) P(to)P(eat)P(healthy)P(food)

Bigram model

N=2

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

Example:

P(I want to eat healthy food)≈ P(I|<start>) P(want|I) P(to|want) P(eat|to) P(healthy |eat) P(food| healthy) P(<end>|food)

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(s) which I had just put into the machine room on the fifth floor is (are) crashing."

Estimating bigram probabilities

- The Maximum Likelihood Estimate

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

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

An example

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

<s> I am Sam </s>

<s> Sam I am </s>

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

$$P(\texttt{I} \mid \texttt{<s>}) = \frac{2}{3} = .67 \qquad P(\texttt{Sam} \mid \texttt{<s>}) = \frac{1}{3} = .33 \qquad P(\texttt{am} \mid \texttt{I}) = \frac{2}{3} = .67$$

$$P(\texttt{</s>} \mid \texttt{Sam}) = \frac{1}{2} = 0.5 \qquad P(\texttt{Sam} \mid \texttt{am}) = \frac{1}{2} = .5 \qquad P(\texttt{do} \mid \texttt{I}) = \frac{1}{3} = .33$$

More examples:
Berkeley Restaurant Project sentences

- can you tell me about any good cantonese restaurants close by
- mid priced thai food is what i'm looking for
- tell me about chez panisse
- 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

Raw bigram counts

- Out of 9222 sentences

	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

Raw bigram probabilities

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

P((i|i)=5/2533=0.002
P(want|i)= 827 / 2533 = 0.33
P(to|i) = 0 / 2533 =0
P(eat|i)= 9 / 2533 =0.0036
P(i|want)=2/927=0.0022
P(to|want) = 608 / 927=0.66
P(eat|to) = 686/2417 =0.28

	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

Bigram estimates of sentence probabilities

innovate achieve lead

P(<s> I want english food </s>) =

P(I|<s>)

× P(want|I)

× P(english|want)

× P(food|english)

× P(</s>|food)

= .000031

What kinds of knowledge?

- P(english|want) = .0011
- P(chinese|want) = .0065
- P(to|want) = .66
- P(eat | to) = .28
- P(food | to) = 0
- P(want | spend) = 0
- P (i | <s>) = .25

Practical Issues

- We do everything in log space
  - Avoid underflow
  - (also adding is faster than multiplying)

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

Google N-Gram Release

AUG
3

All Our N-gram are Belong to You

Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

Here at Google Research we have been using word n-gram models for a variety of R&D projects,

...

That's why we decided to share this enormous dataset with everyone. We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 unique words, after discarding words that appear less than 200 times.

Google N-Gram Release

- 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

http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html

https://pypi.org/project/google-ngram-downloader/

The perils of overfitting

- N-grams only work well for word prediction if the test corpus looks like the training corpus
  - In real life, it often doesn't
  - We need to train robust models that generalize!
  - One kind of generalization: Zeros!
    - Things that don't ever occur in the training set
      - But occur in the test set

Zeros

- Training set:
  … denied the allegations
  … denied the reports
  … denied the claims
  … denied the request

  P(“offer” | denied the) = 0

- Test set
  … denied the offer
  … denied the loan

Zero probability bigrams

- Bigrams with zero probability
  - mean that we will assign 0 probability to the test set!

Laplace Smoothing(Add 1 smoothing)

- Pretend we saw each word one more time than we did
- 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:

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

Maximum Likelihood Estimates

- The maximum likelihood estimate
  - of some parameter of a model M from a training set T
  - maximizes the likelihood of the training set T given the model M
- Suppose the word “bagel” occurs 400 times in a corpus of a million words
- What is the probability that a random word from some other text will be “bagel”?
- MLE estimate is 400/1,000,000 = .0004
- This may be a bad estimate for some other corpus
  - But it is the **estimate** that makes it **most likely** that “bagel” will occur 400 times in a million word corpus.

Raw bigram counts

- Out of 9222 sentences

	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

Raw bigram probabilities

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

P((i|i)=5/2533=0.002
P(want|i)= 827 / 2533 = 0.33
P(to|i) = 0 / 2533 =0
P(eat|i)= 9 / 2533 =0.0036
P(i|want)=2/927=0.0022
P(to|want) = 608 / 927=0.66
P(eat|to) = 686/2417 =0.28

	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

Berkeley Restaurant Corpus: Laplace smoothed bigram counts

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

V = Total number of unique words in corpus = 1446

	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 bigrams

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

V = Total number of unique words in corpus = 1446
P(want|i)= 828 /2533+1446 = 828/3979 = 0.21
P(to|i) = 1/3979 =0.00025
P(eat|i)= 10/3979=0.0025

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

P(to|want) = 609/927+1446=609/2373=0.26
P(eat|to) = 687/2417+1446 =0.18

	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 counts

innovate achieve lead

$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n)+1] \times C(w_{n-1})}{C(w_{n-1})+V}$$

c(i,want) = (828*2533)/ (2533+1446) =527

	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

Compare with raw 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

	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

Add-1 estimation is a blunt instrument

- So add-1 isn't used for N-grams:
  - We'll see better methods
- But add-1 is used to smooth other NLP models
  - For text classification
  - In domains where the number of zeros isn't so huge.

Backoff and Interpolation

- Sometimes it helps to use **less** context
  - Condition on less context for contexts you haven't learned much about
- **Backoff:**
  - use trigram if you have good evidence,
  - otherwise bigram, otherwise unigram
- **Interpolation:**
  - mix unigram, bigram, trigram

- Interpolation works better

Linear Interpolation

- Simple interpolation

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

$$\sum_i \lambda_i = 1$$

- Lambdas conditional on context:

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

How to set 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:

$$\log P(w_1 \ldots w_n \mid M(l_1 \ldots l_k)) = \sum_i \log P_{M(l_1 \ldots l_k)}(w_i \mid w_{i-1})$$

Unknown words: Open versus closed vocabulary tasks

- If we know all the words in advanced
  - Vocabulary V is fixed
  - Closed vocabulary task
- Often we don't know this
  - **Out Of Vocabulary** = OOV words
  - Open vocabulary task
- Instead: create an unknown word token <UNK>
  - Training of <UNK> probabilities
    - Create a fixed lexicon L of size V
    - At text normalization phase, any training word not in L changed to <UNK>
    - Now we train its probabilities like a normal word
  - At decoding time
    - If text input: Use UNK probabilities for any word not in training

Huge web-scale n-grams

- How to deal with, e.g., Google N-gram corpus
- Pruning
  - Only store N-grams with count > threshold.
    - Remove singletons of higher-order n-grams

Smoothing for Web-scale N-grams

- "Stupid backoff" (Brants *et al*. 2007)
- No discounting, just use relative frequencies

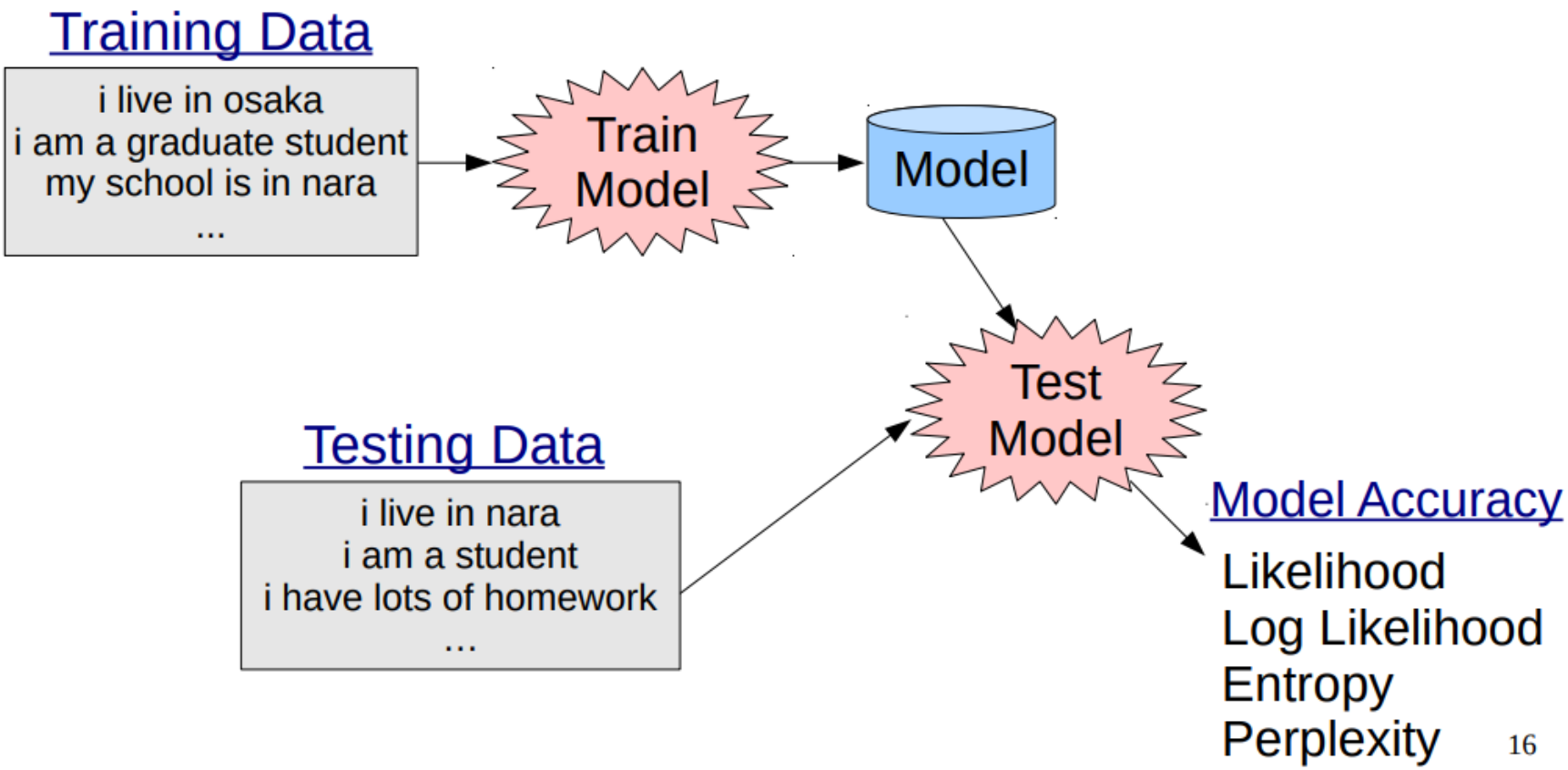
$$S(w_i \mid w_{i-k+1}^{i-1}) = \begin{cases} \dfrac{\text{count}(w_{i-k+1}^{i})}{\text{count}(w_{i-k+1}^{i-1})} & \text{if } \text{count}(w_{i-k+1}^{i}) > 0 \\ 0.4\,S(w_i \mid w_{i-k+2}^{i-1}) & \text{otherwise} \end{cases}$$

$$S(w_i) = \frac{\text{count}(w_i)}{N}$$

Evaluation: How good is our 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 haven’t 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.

Experimental setup



Extrinsic evaluation of N-gram models

- Best evaluation for comparing models A and B
  - 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

Difficulty of extrinsic (in-vivo) evaluation of N-gram models

- Extrinsic evaluation
  - Time-consuming; can take days or weeks
- So
  - Sometimes use **intrinsic** evaluation: **perplexity**
  - Bad approximation
    - unless the test data looks **just** like the training data
    - So **generally only useful in pilot experiments**
  - But is helpful to think about.

Intuition of Perplexity

- The Shannon Game:
  - How well can we 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?)

- A better model of a text
  - is one which assigns a higher probability to the word that actually occurs

Why Perplexity

- **Longer sentences always have smaller probabilities**
- Sentence probability is computed as:
- $P(w_1\, w_2 \ldots \ldots w_n) = \prod_{k=1}^{n} P(w_k \,|\, w_1^{k-1})$
- Multiplying many probabilities (each < 1) gives a **very tiny number**.
- A **longer sentence** will always have a smaller probability than a short sentence — even if the model is good.

So comparing raw probabilities is meaningless.

- Example:
  - Model A:
    P("I like cats") = 0.01
    P("I like cute cats very much") = 0.00003
  - Does this mean model A is bad?
    No — the second sentence is just longer.

Why Perplexity

- Two models might give:
  - Model A: probability = 0.0001
  - Model B: probability = 0.000000002
- If the sentences are different lengths, **you can't compare them directly**.
- We need a **length-normalized score**.
- It converts the probability of the whole sentence into an **average per-word confusion measure**:

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

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

- This is the **geometric mean of inverse probabilities**.

Minimizing perplexity is the same as maximizing probability

Why Perplexity

- **Length Normalized**

Perplexity makes models **comparable even if sentences differ in length**.
It answers:
“On average, how many choices does the model seem confused between per word?”

- **Easy to interpret**

If perplexity = **10**,
→ The model is as uncertain as choosing from **10 equally likely words**.
If perplexity = **2**,
→ Very confident (like flipping a coin).
If perplexity = **100**,
→ Very confused.

This interpretation is **much clearer** than reading probabilities like 0.00000003265

Why Perplexity

- **Allows fair model comparison**

Two LMs can be compared fairly

Language modeling benchmarks use PPL for standardized evaluation

Perplexity works across n-grams, RNNs, LSTMs, transformers

- **Stable and robust metric**

Probabilities shrink exponentially.
Perplexity stays within a **reasonable numeric range** (e.g., 10–400), making comparison easy

Example

Sentence: **"I love NLP models"**
Number of words: **N = 4**
Suppose a language model gives the following probabilities:

Word	Probability
I	0.10
love	0.20
NLP	0.25
models	0.50

Compute total sentence probability
P=0.10×0.20×0.25×0.50
P=0.0025

Apply the perplexity formula:

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

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

N=4
PPL=$(0.0025)^{-1/4}$
PPL = $(1/400)^{-1/4}$ ≈4.47

Model is as uncertain as choosing between **about 5 equally likely words** at each position.
Lower perplexity = better language model.

Perplexity as branching factor

- Let's suppose a sentence consisting of random digits 0 to 9 Ex: 0,2,1,3……N
- What is the perplexity of this sentence according to a model that assign P=1/10 to each digit?

$$\begin{aligned} \mathrm{PP}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\ &= \left(\frac{1}{10}^{N}\right)^{-\frac{1}{N}} \\ &= \frac{1}{10}^{-1} \\ &= 10 \end{aligned}$$

Lower perplexity = better model

- Training 38 million words, test 1.5 million words, WSJ

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Corpora of text and speech

- Brown Corpus
- Wall Street Journal
- AP newswire
- Hansards
- DARPA/NIST text/speech corpora (Call Home, ATIS, switchboard, Broadcast News, TDT, Communicator)
- TRAINS, Radio News

N-gram versus LLM(neural language models)

Feature	N-gram Models	LLMs
Training Data Needed	Very small	Huge
Compute	Low	Very high
Interpretability	Excellent	Poor
Handles Long Context?	No	Yes
Quality of Language?	Low–medium	Very high
Hallucination	Almost none	Possible
Deployment	Easy, lightweight	Heavy, expensive
Use Cases	Small, fast, interpretable systems	Large-scale NLP tasks

References

https://books.google.com/ngrams

https://ai.googleblog.com/2006/08/all-our-n-gram-are-belong-to-you.html

http://www.speech.sri.com/projects/srilm/

https://web.stanford.edu/~jurafsky/NLPCourseraSlides.html

https://www.kaggle.com/code/alvations/n-gram-language-model-with-nltk/notebook

https://levelup.gitconnected.com/bi-tri-and-n-grams-with-python-a9717264e6f2

Dr. Chetana Gavankar has over 27 years of Teaching, Research and Industry experience. She has published papers in peer reviewed international conferences and journals. She is also reviewer for multiple conferences and journals. She has worked on different projects with multiple industries and received awards for her research work . Her areas of research interests include Natural Language Processing, Information Retrieval, Web Mining and Semantic Web, Ontology, Big Data Analytics, Machine learning, Deep learning and Artificial Intelligence.

Thank you!!