Natural Language Processing

Assist. Prof. Dr. Tuğba YILDIZ

İSTANBUL BİLGİ UNIVERSITY
Department of Computer Engineering

March 8, 2019

Language Modeling

- "Please turn your homework..."
- we will conclude the sentence
- which one is available with **in** or **over** or **the**
- **in** or **over** not **the**

Language Modeling

- guessing the next word **(or word prediction)** is important
- word prediction is an essential subtask for :
  - speech recognition
  - hand-writing recognition
  - augmentative communication (güçlendirici iletişim) for the disabled
  - spelling error detection
  - machine translation

Language Modeling

- in such tasks;
- word identification is difficult because the input is very noisy and ambiguous
- thus looking at **previous words** can give us an important cue about what the **next ones** are going to be.

Language Modeling

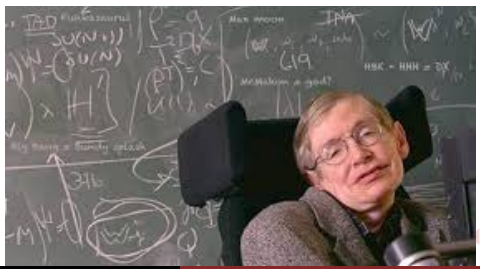
- **speech recognition:**
  - input speech sounds are very confusable
  - many words sound extremely similar
  - P(I saw a van) $-->$ P (eyes awe of an)

Language Modeling

- **handwriting recognition:** how probabilities of word sequences can help
  - "Take the Money and Run" (Woody Allen)
  - Woody Allen tries to rob a bank with sloppily written hold-up note
  - the teller incorrectly reads as "I have a gub"
  - "I have a gun" is more probable than "I have a gub" or "I have a gull".
  - https://www.youtube.com/watch?v=F8iYrG7gms8

Language Modeling

- **augmentative communication for the disabled**:
  - these are computer systems that help the disabled in communication
  - people who are unable to use speech or sign-language to communicate, like the physicist **Steven Hawkings**
  - use systems that speak for them, letting them choose words with simple hand movements
  - either by spelling them out, or by selecting from a menu of possible words
  - But spelling is very slow, and a menu of words obviously can't have all possible English words on one screen.
  - Thus it is important to be able to know which words the speaker is likely to want to use next, so as to put those on the menu.



Language Modeling

- **Spelling-correction :** need to find and correct spelling errors
- "They are leaving in about fifteen **minuets** to go to her house"
- we can not find them bu just flagging words that are not in dictionary
- spell checker can use a probability estimator to detect errors and suggest higher probability corrections.

Language Modeling

- **Machine translation:**

Fig. 2 a sentence in Chinese

- he briefed to reporters on the chief contents of the statement
- he briefed reporters on the chief contents of the statement
- he briefed to reporters on the main contents of the statment
- he briefed reporters on the main contents of the statment
- select the highest probability

Language Modeling

- **Other tasks:**
  - POS tagging
  - NLG
  - Word similarity
  - Authorship identification

Language Modeling

- Finding the probability of a sentence or a sequence of words
  - P(S) = P($w_1, w_2, w_3, ..., w_n$)
- We formalize word prediction with probabilistic models classed **N-gram models**
- N-gram models predict the next word from previous N-1 words
- such statistical models of word sequences are also called **language models** or **LMs**.
- computing probability of next word is related to computing the probability of a sequence of words.

Corpus

- Probabilities are based on counting things
- A corpus is a computer-readable collection of text or speech
  - Corpus of Contemporary American English
  - The British National Corpus
  - The International Corpus of English
  - The Google N-gram Corpus (https://books.google.com/ngrams)
  - But also many small corpora for particular domains/tasks...

Counting Words in Corpora

- Probabilities are based on counting things
- **corpus (plural corpora)** is on-line collection of text or speech
- Brown Corpus: 1 million word collection samples from 500 written text from different genres(newspaper, novels, non-fiction, academics, etc.)
- "he stepped out into the hall, was delighted to encounter a water brother."
- 13 words without punctuation marks
- 15 words with punctuation marks

Counting Words in Corpora

- punctuation is critical:
  - finding boundaries of things (commas, periods, colons, vs. -csv)
  - identifying some aspects of meaning (question marks, quotation marks, etc.)
  - POS tagging or speech synthesis

Counting Words in Corpora

- cat vs cats
  - two words have the same **lemma** but different **wordforms**
  - **lemma** is a set of lexical forms having the same stem, the same major POS and same word-sense
  - **wordform** is the full inflected or derived form of the word
    - Imagination is an example of a word that has all four wordforms-noun, verb, adjective and adverb
    - Imagination (noun), imagined (verb), imaginative (adj), imaginatively (adverb)

Counting Words in Corpora

- type: number of distinct words in corpus or vocabulary size V
- token: total number N of running words
- "They picnicked by the pool, then lay back on the grass and looked at the stars."
- type: 14
- token: 16

Word occurrence

- Google N-Gram corpus:
  - 1,024,908,267,229 word tokens
  - 13,588,391 word types
- Large English dictionaries have around 500k word types

Word Frequency

Rank	Word	Count	Freq(%)
1	The	69970	6.8872
2	of	36410	3.5839
3	and	28854	2.8401
4	to	26154	2.5744
5	a	23363	2.2996
6	in	21345	2.1010
7	that	10594	1.0428
8	is	10102	0.9943
9	was	9815	0.9661
10	He	9542	0.9392
11	for	9489	0.9340
12	it	8760	0.8623
13	with	7290	0.7176
14	as	7251	0.7137
15	his	6996	0.6886
16	on	6742	0.6636
17	be	6376	0.6276
18	at	5377	0.5293
19	by	5307	0.5224
20	I	5180	0.5099

Fig.3 Frequency table

Zipf's Law

- The frequency of any word is inversely proportional to its rank in the frequency table
- Given a corpus of natural language utterances, the most frequent word will occur approximately
  - twice as often as the second most frequent word,
  - three times as often as the third most frequent word,
  - ...
- Rank of a word times its frequency is approximately a constant
  - Rank $\cdot$ Freq $\approx$ c
  - c $\approx$ 0.1 for English

Zipf's Law

Rank	Word	Count	Freq(%)	Freq x Rank
1	The	69970	6.8872	0.06887
2	of	36410	3.5839	0.07167
3	and	28854	2.8401	0.08520
4	to	26154	2.5744	0.10297
5	a	23363	2.2996	0.11498
6	in	21345	2.1010	0.12606
7	that	10594	1.0428	0.07299
8	is	10102	0.9943	0.07954
9	was	9815	0.9661	0.08694
10	He	9542	0.9392	0.09392
11	for	9489	0.9340	0.10274
12	it	8760	0.8623	0.10347
13	with	7290	0.7176	0.09328
14	as	7251	0.7137	0.09991
15	his	6996	0.6886	0.10329
16	on	6742	0.6636	0.10617
17	be	6376	0.6276	0.10669
18	at	5377	0.5293	0.09527
19	by	5307	0.5224	0.09925
20	I	5180	0.5099	0.10198

Zipf's Law

- Zipf's Law is not very accurate for very frequent and very infrequent words

Rank	Word	Count	Freq(%)	Freq x Rank
1	The	69970	6.8872	0.06887
2	of	36410	3.5839	0.07167
3	and	28854	2.8401	0.08520
4	to	26154	2.5744	0.10297
5	a	23363	2.2996	0.11498

Fig.5 Frequency table for frequent words

Zipf's Law

- But very precise for intermediate ranks

Rank	Word	Count	Freq(%)	Freq x Rank
1000	current	104	0.0102	0.10200
1001	spent	104	0.0102	0.10210
1002	eight	104	0.0102	0.10220
1003	covered	104	0.0102	0.10230
1004	Negro	104	0.0102	0.10240
1005	role	104	0.0102	0.10251
1006	played	104	0.0102	0.10261
1007	I'd	104	0.0102	0.10271
1008	date	103	0.0101	0.10180
1009	council	103	0.0101	0.10190
1010	race	103	0.0101	0.10201

Fig.6 Frequency table for intermediate rank

Language Models

- ways to assign probabilities to strings of words
- whether for computing the probability of an entire sentence or for giving a probabilistic prediction of what the next word will be in a sequence.
- in simplest model, any word of the language follow any other word!
- in the probabilistic version of this theory, every word would have an equal probability of following every other word.
- if English had 100,000 words, the probability of any word following any other word would be 0.00001.

Language Models

- the word **the** has a high relative frequency
- it occurs 69,971 times in the Brown corpus of 1,000,000 words (i.e. 7% of the words in this particular corpus are **the**).
- By contrast the word **rabbit** occurs only 11 times in the Brown corpus (0.00001).
- We can use these **relative frequencies** to assign a probability distribution across following words.

Language Models

- "Just then, the white..."
- in this context, **rabbit** seems like a more reasonable word to follow **white** than **the** does
- in this scenario, instead of just looking at the **individual relative frequencies** of words, we should look at the **conditional probability** of a word given the previous words

Language Models

- **Goal:** to compute the probability of sentence or sequence of words
- P$(w_1, w_2, w_3, w_4, w_5 .... w_n)$
- **Related task:** probability of an upcoming word
- P$(w_5|w_1, w_2, w_3, w_4)$
- a model computes the either of these:
- P(w)
- P$(w_n|w_1, w_2, w_3 ... w_{n-1})$
- called "Language Modeling (LM)"

Language Models

- if we want to know the **joint probability** of a an entire sequence of words like "its water is so transparent", we could do:
- "out of all possible sequences of 5 words how many of them are **its water is so transparent**?"
- get the count of **its water is so transparent**
- divide by the sum of counts of all possible 5 word sequences
- that seems rather a lot to estimate!
- for this reason, we need to introduce cleverer ways of estimating the probability of word w given history h or probability of an entire word sequence

Language Models

- to represent the probability of a particular random variable X_i taking on the value "the"
- P(X_i="the") : P(the)
- sequence of N words: $w_1...w_n$ or w_1^n
- for joint probability of each word in sequence, P $(X_1 = w_1, X_2 = w_2, X_3 = w_3, ...)$: P$(w_1, w_2, ..., w_n)$
- P$(w_1, w_2, ..., w_n)$: P(w_1^n)
- how can we compute probabilities of entire sequences like P$(w_1, w_2, ...w_n)$?

Language Models

- compute the probability P of a word w given some history h or $P(w|h)$
- history: "its water is so transparent that"
- how to compute joint probability
- P(its, water, is, so, transparent, that)
- **CHAIN RULE!**

Simple N-gram

- to decompose this probability using **chain rule of probability**

$$\begin{aligned} P(X_1 \ldots X_n) &= P(X_1)P(X_2|X_1)P(X_3|X_1^2)\ldots P(X_n|X_1^{n-1}) \\ &= \prod_{k=1}^{n} P(X_k|X_1^{k-1}) \end{aligned}$$

Fig.7 Chain rule

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

Fig.8 Applying chain rule

Simple N-gram

- More variables:
- P(A,B,C,D)=P(A)P(B|A)P(C|A,B)P(D|A,B,C)
- P(“its water is so transparent”) = P(its) x P(water | its) x P(is | its water) x P(so | its water is) x P(transparent | its water is so)

Chain Rule

- The chain rule shows the link between computing the joint probability of a sequence and computing the conditional probability of a word given previous words
- P(w1 w2 w3 w4) = P(w_1)P($w_2|w_1$)P($w_3|w_1^2$)P($w_4|w_1^3$)
- we could estimate the **joint probability** of an entire sequence of words by multiplying together a number of **conditional probabilities**.

Back to the conditionla prob.

$$P(S)=P(w_1)\cdot P(w_2|w_1)\cdot P(w_3|w_1,w_2)\dots P(w_n|w_1,w_2,,w_3,\dots,,w_n)$$

$$P(S)=\prod_i^n P(w_i|w_1,w_2,\dots,w_{i-1})$$

P(Computer,can,recognize,speech) = P(Computer)·
P(can|Computer)·
P(recognize|Computer can)·
P(speech|Computer can recognize)

Fig.9 Conditional probability

Maximum Likelihood Estimation

- P(speech|Computer can recognize)

$$P(speech|Computer\ can\ recognize) = \frac{\#(Computer\ can\ recognize\ speech)}{\#(Computer\ can\ recognize)}$$

Fig.10 MLE

- Too many phrases
- Limited text for estimating probabilities
- Simplification assumption

Simple N-gram

- but using the chain rule does not really seem to help us!
- how can we compute probabilities like P $(w_n|w_1^{n-1})$?
- we do not know any way to compute the exact probability of a word given a long sequence of preceding words, P$(w_n|w_1^{n-1})$
- we can not just count the number of times every word occurs following every long string
- we would need far too large a corpus
- we can not just estimate by counting the number of times every word occurs following every long string
- because language is creative and any particular context might have never occurred before!

Markov assumption

$$P(S) = \prod_{i-1}^{n} P(w_i | w_1, w_2, \ldots, w_{i-1})$$

$$P(S) = \prod_{i-1}^{n} P(w_i | w_{i-1})$$

Markov assumption

P(Computer,can,recognize,speech) = P(Computer)·
P(can|Computer)·
P(recognize|Computer can)·
P(speech|Computer can recognize)

P(Computer,can,recognize,speech) = P(Computer)·
P(can|Computer)·
P(recognize|can)·
P(speech|recognize)

$$P(speech|recognize) = \frac{\#(recognize\ speech)}{\#(recognize)}$$

Simple N-gram

- Unigram: $P(S) = \prod_{i-1}^{n} P(w_i)$
- Bigram: $P(S) = \prod_{i-1}^{n} P(w_i | w_{i-1})$
- Trigram: $P(S) = \prod_{i-1}^{n} P(w_i | w_{i-1}, w_{i-2})$
- N-gram: $P(S) = \prod_{i-1}^{n} P(w_i | w_1, w_2, \ldots, w_{i-1})$

Simple N-gram

- instead of computing the probability of a word given its entire history, we will **approximate** the history by just the last few words.
- Simplest case: uni-gram model
- not enough!
- $\mathrm{P}(w_1, w_2, ...w_n) \approx \prod_i \mathrm{P}(w_i)$
- fifth, an, of, futures, the , an...
- **the bi-gram model**, for example, approximates the probability of a word given all the previous words $\mathrm{P}(w_n|w_1^{n-1})$ by using only the conditional probability of the preceding word $\mathrm{P}(w_n|w_{n-1})$

Simple N-gram

- P(the | Walden Pond's water is so transparent that)
- we approximate it with the probability P(the | that)
- P(rabbit | Just the other I day I saw a)
- P(rabbit | a)
- when we use a bi-gram model to predict the conditional probability of the next word we are making following approximation
- $P(w_n|w_1^{n-1} \approx) P(w_n|w_{n-1})$
- This assumption that the probability of a word depends only on the previous word is called **Markov assumption**.

Simple N-gram

- Simplifying assumption:
- P(the | its water is so transparent that) $\approx$ P(the | that)
- or maybe:
- P(the | its water is so transparent that) $\approx$ P(the | transparent that)

Simple N-gram

- Simplest case: uni-gram model
- $\mathrm{P}(w_1, w_2, ...w_n) \approx \prod_i \mathrm{P}(w_i)$
- fifth, an, of, futures, the , an...

Simple N-gram

- bi-gram models:
  - condition on the previous word
  - $P(w_i|w_1, w_2, ...w_{i-1}) \approx P(w_i|w_{i-1})$
  - car, parking, lot, of, the agreement,...
- N-gram models:
  - we can extend to tri-grams, four-grams, ...

Simple N-gram

- **Markov models** are the class of probabilistic models that assume that we can predict the probability of some future unit without looking too far into the past.
- a **Markov chain** is a kind of weighted finite-state automaton
- the intuition of the term Markov in Markov chain is that the next state of a weighted FSA is always dependent on a finite history
- The simple bi-gram model can be viewed as a simple kind of Markov chain which has one state for each word.

Simple N-gram

- we can generalized the bi-gram to trigram(which looks two words into past)
- thus to N-gram (which looks N-1 words into the past)
- $\mathrm{P}(w_n|w_1^{n-1}) \approx \mathrm{P}(w_n|w_{n-N+1}^{n-1})$
- a bi-gram is called a first-order Markov model (because it looks one token into the past)
- a trigram is a second-order Markov model
- and in general an N-gram is a N-1th order Markov model.

Simple N-gram

- Given the bi-gram assumption for the probability of an individual word, we can compute the probability of a word sequence by substituting
- $\mathrm{P}(w_1^n) \approx \prod_{k=1}^{n} \mathrm{P}(w_k|w_{k-1})$
- How do we estimate these bi-gram or N-gram probabilities?
- The simplest and most intuitive way to estimate probabilities is called **Maximum Likelihood Estimation, or MLE**
- We get the MLE estimate for the parameters of an N-gram model by taking counts from a corpus, and normalizing them so they lie between 0 and 1

Simple N-gram

- the sum of all bi-gram counts that start with a given word w_{n-1} must be equal to the uni-gram count for that word w_{n-1} .

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

Simple N-gram

- mini-corpus of three sentences:
- $< s >$ I am Sam $< /s >$
- $< s >$ Sam I am $< /s >$
- $< s >$ I do not like green eggs and ham $< /s >$

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

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

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

Simple N-gram

- <s> I saw the boy </s>
- <s> the man is working </s>
- <s> I walked in the street </s>

- Vocabulary:
  - V = {I,saw,the,boy,man,is,working,walked,in,street}

Simple N-gram

- <s> I saw the boy </s>
- <s> the man is working </s>
- <s> I walked in the street </s

boy	I	in	is	man	saw	street	the	walked	working
1	2	1	1	1	1	1	3	1	1

	boy	I	in	is	man	saw	street	the	walked	working
boy	0	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	1	0	0	1	0
in	0	0	0	0	0	0	0	1	0	0
is	0	0	0	0	0	0	0	0	0	1
man	0	0	0	1	0	0	0	0	0	0
saw	0	0	0	0	0	0	0	1	0	0
street	0	0	0	0	0	0	0	0	0	0
the	1	0	0	0	1	0	1	0	0	0
walked	0	0	1	0	0	0	0	0	0	0
working	0	0	0	0	0	0	0	0	0	0

Simple N-gram

- Estimation of maximum likelihood for a new sentence
  - <s> I saw the man </s>

$$P(S)=P(I|<s>)\cdot P(saw|I)\cdot P(the|saw)\cdot P(man|the)$$

$$P(S)=\frac{\#(<s>I)}{\#(<s>)}\cdot\frac{\#(I\,saw)}{\#(I)}\cdot\frac{\#(saw\,the)}{\#(saw)}\cdot\frac{\#(the\,man)}{\#(the)}$$

$$P(S)=\frac{2}{3}\cdot\frac{1}{2}\cdot\frac{1}{1}\cdot\frac{1}{3}$$

Simple N-gram

- Example from Berkeley Restaurant Project, a dialogue system
  - I'm looking for Cantonese food.
  - I'd like to eat dinner someplace nearby.
  - 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.
  - I definitely do not want to have cheap Chinese food.
  - When is Caffe Venezia open during the day?
  - I don't wanna walk more than ten minutes.

Simple N-gram

- a sample of the bi-gram probabilities for some of the words that can follow the word eat, taken from actual sentences spoken by users

eat on	.16	eat Thai	.03
eat some	.06	eat breakfast	.03
eat lunch	.06	eat in	.02
eat dinner	.05	eat Chinese	.02
eat at	.04	eat Mexican	.02
eat a	.04	eat tomorrow	.01
eat Indian	.04	eat dessert	.007
eat today	.03	eat British	.001

Fig 5. bi-grams of the words followed by eat

Simple N-gram

- a sample of the bi-gram probabilities for some of the words that can follow the word eat, taken from actual sentences spoken by users

<s> I	.25	I want	.32	want to	.65	to eat	.26	British food	.60
<s> I'd	.06	I would	.29	want a	.05	to have	.14	British restaurant	.15
<s> Tell	.04	I don't	.08	want some	.04	to spend	.09	British cuisine	.01
<s> I'm	.02	I have	.04	want thai	.01	to be	.02	British lunch	.01

Fig 6. more fragments from bi-grams

Simple N-gram

- compute the probability of sentences like
- "I want to eat British food or I want to eat Chinese food"
- P(I want to eat British food) = P(I | $<$ s $>$)P(want|I)P(to|want)P(eat|to)P(British|eat)P(food|British)
- .25 * .32 * .65 * .26 * .002 *.60
- .000016

Simple N-gram

- A trigram model looks just the same as a bi-gram model, except that we condition on the two previous words
- (e.g. P(food | eat British)
- To compute trigram probabilities at the very beginning of sentence, we can use two pseudo-words for the first trigram
- (e.g P(I | $< s >< s >$)).

Simple N-gram

- Example from Berkeley Restaurant Project, a dialogue system

	I	want	to	eat	Chinese	food	lunch
I	8	1087	0	13	0	0	0
want	3	0	786	0	6	8	6
to	3	0	10	860	3	0	12
eat	0	0	2	0	19	2	52
Chinese	2	0	0	0	0	120	1
food	19	0	17	0	0	0	0
lunch	4	0	0	0	0	1	0

Fig 7. bi-grams counts of 7 of the words (out of V = 1616) in the Berkeley Restaurant Project corpus of ∼10000 sentences

- majority of the values are zero

Simple N-gram

I	3437
want	1215
to	3256
eat	938
Chinese	213
food	1506
lunch	459

Fig 8. uni-grams counts of 7 words

Simple N-gram

	I	want	to	eat	Chinese	food	lunch
I	.0023	.32	0	.0038	0	0	0
want	.0025	0	.65	0	.0049	.0066	.0049
to	.00092	0	.0031	.26	.00092	0	.0037
eat	0	0	.0021	0	.020	.0021	.055
Chinese	.0094	0	0	0	0	.56	.0047
food	.013	0	.011	0	0	0	0
lunch	.0087	0	0	0	0	.0022	0

Fig 9. bi-gram probabilities for 7 words

Simple N-gram

- If we are computing the probability of a very long string (like a paragraph or an entire document), it is more customary to do the computation in log space; we take the log of each probability (the logprob)
- add all the logs (since adding in log space is equivalent to multiplying in linear space)
- For this reason many standard programs for computing N-grams actually store and calculate all probabilities as logprobs.

N-grams and their sensitivity

1. the first is the increasing accuracy of N-gram models as we increase the value of N.
2. the second is their very strong dependency on their training corpus (in particular its genre and its size in words).

N-grams and their sensitivity

- Generating random sentences from different N-gram models.
- It is simplest to visualize how this works for the uni-gram case.
- We continue choosing random numbers and generating words until we randomly generate the sentence-final token $< /s >$.
- The same technique can be used to generate bi-grams by first generating a random bi-gram that starts with $< s >$ (according to its bi-gram prob- ability), then choosing a random bi-gram to follow it (again, according to its conditional probability), and so on.

N-gram Sensitivity to the Training Corpus

Unigram	• To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have • Every enter now severally so, let • Hill he late speaks; or! a more to leg less first you enter • Are where exeunt and sighs have rise excellency took of.. Sleep knave we. near; vile like
Bigram	• What means, sir. I confess she? then all sorts, he is trim, captain. •Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. •What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman? •Enter Menenius, if it so many good direction found'st thou art a strong upon command of fear not a liberal largess given away, Falstaff! Exeunt
Trigram	• Sweet prince, Falstaff shall die. Harry of Monmouth's grave. • This shall forbid it should be branded, if renown made it empty. • Indeed the duke; and had a very good friend. • Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.
Quadrigram	• King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; • Will you not tell me who I am? • It cannot be but so. • Indeed the short and the long. Marry, 'tis a noble Lepidus.

N-grams and their sensitivity

- The longer the context on which we train the model, the more coherent the sentences.
- In the uni-gram sentences, there is no coherent relation between words, nor any sentence-final punctuation.
- The bi-gram sentences have some very local word-to-word coherence (especially if we consider that punctuation counts as a word).
- The tri-gram and quadri-gram (four-gram) sentences are beginning to look a lot like Shakespeare.
- quadri-gram sentences shows that they look a little too much like Shakespeare.
- The words "It cannot be but so" are directly from **King John**.

N-gram Sensitivity to the Training Corpus

unigram: Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives
bigram: 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
trigram: 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

Fig.11 Randomly generated sentences from three N-gram models computed from Wall Street Journal (40 million words)

Training and Test Set

- very strong dependency on their training corpus (in particular its genre and its size in words).
- we can formalize this idea of training on some data, and testing on some other data
- these two data sets as a training set and a test set (or a training corpus and a test corpus).
- we train the statistical parameters of the model on the training set
- then use this trained model to compute probabilities on the test set
- A test set is an unseen dataset that is different from our training set, totally unused

Unknown Words: Open versus closed vocabulary tasks

- the closed vocabulary assumption is the assumption that we have such a lexicon, and that the test set can only contain words from this lexicon.
- the closed vocabulary task thus assumes there are no unknown words.
- but of course this is a simplification
- unseen events unknown words, or out of vocabulary (OOV) words.
- the percentage of OOV words that appear in the test set is called the OOV rate.

Simple N-gram

- Estimation of maximum likelihood for a new sentence
  - <s> I saw the man </s>

$$P(S)=P(I|<s>)\cdot P(saw|I)\cdot P(the|saw)\cdot P(man|the)$$

$$P(S)=\frac{\#(<s>I)}{\#(<s>)}\cdot\frac{\#(I\,saw)}{\#(I)}\cdot\frac{\#(saw\,the)}{\#(saw)}\cdot\frac{\#(the\,man)}{\#(the)}$$

$$P(S)=\frac{2}{3}\cdot\frac{1}{2}\cdot\frac{1}{1}\cdot\frac{1}{3}$$

Unknown Words: Open versus closed vocabulary tasks

- an open vocabulary system is one where we model these potential unknown words in the test set by adding a pseudo-word called <UNK>.
- we can train the probabilities of the unknown word model <UNK> as follows:
  1. choose a vocabulary (word list) which is fixed in advance
  2. convert in the training set any word that is not in this set (OOV) to unknown word (UNK) in a text normalization step
  3. estimate the probabilities for <UNK> from its counts just like any other regular word in the training set

Simple N-gram

- Corpus:
  - <s> I saw the boy </s>
  - <s> the man is working </s>
  - <s> I walked in the street </s>
- <s> I saw the man in the street </s>

$$P(S)=P(I)\cdot P(saw|I)\cdot P(the|saw)\cdot P(man|the)\cdot P(in|man)\cdot P(the|in)\cdot P(street|the)$$

$$P(S)=\frac{\#(I)}{\#(<s>)}\cdot\frac{\#(I\,saw)}{\#(I)}\cdot\frac{\#(saw\,the)}{\#(saw)}\cdot\frac{\#(the\,man)}{\#(the)}\cdot\frac{\#(man\,in)}{\#(man)}\cdot\frac{\#(in\,the)}{\#(in)}\cdot\frac{\#(the\,street)}{\#(the)}$$

$$P(S)=\frac{2}{3}\cdot\frac{1}{2}\cdot\frac{1}{1}\cdot\frac{1}{3}\cdot\frac{0}{1}\cdot\frac{1}{1}\cdot\frac{1}{3}$$

Smooting

- the bi-gram matrix for any given training corpus is sparse
- it is bound to have a very large number of cases of putative "zero probability bi-grams"
- This task of reevaluating some of the zero-probability and low-probability N-grams, and assigning them non-zero values, is called smoothing.

Smooting

- Training set:
  - ...denied the reports
  - ...denied the claims
  - ...denied the request
- Test set:
  - ...denied the offer
  - ...denied the loan
- P("offer" | denied the) = 0

Smooting

- Add-one smooting (Laplace smooting):
  - just add one to all the counts!

Smooting

- one simple way to do smoothing might be just to take our matrix of bi-gram counts, before we normalize them into probabilities
- add one to all the counts
- this algorithm is called add-one smoothing.

	I	want	to	eat	Chinese	food	lunch
I	9	1088	1	14	1	1	1
want	4	1	787	1	7	9	7
to	4	1	11	861	4	1	13
eat	1	1	3	1	20	3	53
Chinese	3	1	1	1	1	121	2
food	20	1	18	1	1	1	1
lunch	5	1	1	1	1	2	1

Fig.12 Addone smooting for 7 words

Smooting

I	3437+1616	=	5053
want	1215+1616	=	2931
to	3256+1616	=	4872
eat	938+1616	=	2554
Chinese	213+1616	=	1829
food	1506+1616	=	3122
lunch	459+1616	=	2075

Fig.13 probabilities of addone smooting for 7 words

Smooting

	I	want	to	eat	Chinese	food	lunch
I	.0018	.22	.00020	.0028	.00020	.00020	.00020
want	.0014	.00035	.28	.00035	.0025	.0032	.0025
to	.00082	.00021	.0023	.18	.00082	.00021	.0027
eat	.00039	.00039	.0012	.00039	.0078	.0012	.021
Chinese	.0016	.00055	.00055	.00055	.00055	.066	.0011
food	.0064	.00032	.0058	.00032	.00032	.00032	.00032
lunch	.0024	.00048	.00048	.00048	.00048	.00096	.00048

Fig.14 probabilities of addone smooting for 7 words

Smooting

MLE estimate:

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

Add-1 estimate:

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

Fig.15 probabilities of addone smooting for 7 words

Smooting

- C(want to) changed from 786 to 331!
- We can see this in probability space as well: P (to | want) decreases from .65 in the unsmoothed case to .28 in the smoothed case
- We could avoid this problem by adding smaller values to the counts ('add-one-half' 'add-one-thousandth')
- Although this algorithm does not perform well and is not commonly used, it introduces many of the concepts that we will see in other smoothing algorithms, and also gives us a useful baseline.

Simple N-gram

- Corpus:
  - <s> I saw the boy </s>
  - <s> the man is working </s>
  - <s> I walked in the street </s>
- <s> I saw the man in the street </s>

$$P(S)=P(I)\cdot P(saw|I)\cdot P(the|saw)\cdot P(man|the)\cdot P(in|man)\cdot P(the|in)\cdot P(street|the)$$

$$P(S)=\frac{\#(I)}{\#(<s>)}\cdot\frac{\#(I\,saw)}{\#(I)}\cdot\frac{\#(saw\,the)}{\#(saw)}\cdot\frac{\#(the\,man)}{\#(the)}\cdot\frac{\#(man\,in)}{\#(man)}\cdot\frac{\#(in\,the)}{\#(in)}\cdot\frac{\#(the\,street)}{\#(the)}$$

$$P(S)=\frac{2}{3}\cdot\frac{1}{2}\cdot\frac{1}{1}\cdot\frac{1}{3}\cdot\frac{0}{1}\cdot\frac{1}{1}\cdot\frac{1}{3}$$

Simple N-gram

- Small probability to all unseen n-grams
  - Add one to all counts

	boy	I	in	is	man	saw	street	the	walked	working
boy	1	1	1	1	1	1	1	1	1	1
I	1	1	1	1	1	2	1	1	2	1
in	1	1	1	1	1	1	1	2	1	1
is	1	1	1	1	1	1	1	1	1	2
man	1	1	1	2	1	1	1	1	1	1
saw	1	1	1	1	1	1	1	2	1	1
street	1	1	1	1	1	1	1	1	1	1
the	2	1	1	1	2	1	2	1	1	1
walked	1	1	2	1	1	1	1	1	1	1
working	1	1	1	1	1	1	1	1	1	1

$$P(w_i|w_{i-1}) = \frac{\#(w_{i-1}, w_i)}{\#(w_{i-1})} \longrightarrow P(w_i|w_{i-1}) = \frac{\#(w_{i-1}, w_i)+1}{\#(w_{i-1})+V}$$

Smooting

- Good-Turing
- Kneser-Ney
- Witten-Bell

Smooting

- Practice N-gram in NLTK!

Good example

- Google N-gram!

References

- Speech and Language Processing (3rd ed. draft) by D. Jurafsky & J. H. Martin (web.stanford.edu)