Lecture 7: N-gram Language Models, Processing Web Resources

Ling 1330/2330 Intro to Computational Linguistics

Na-Rae Han, 9/19/2023

Objectives

- Review HW#2 Bigram Speak
  - Producing bigram dictionaries from large corpora
  - A bigram-based statistical language *generation* model
- n-gram language model
  - Estimating sentence probabilities
- N-gram resources
  - Norvig/Google 1T data

Check your NLTK version!

```
>>> import nltk
>>> nltk.__version__
    '3.8.1'
>>>
```

DOUBLE underscores

- Version 3.8.1 is the latest.
- If you have 3.7, you will get different tokenization results.

    🡐 UPGRADE to the latest version! See me/Tianyi.

Homework #2: what you achieved

- You computed basic stats (type & token counts) of:
  - The Bible
  - Jane Austen's 3 novels
- You produced bigram data objects of the two corpora
- You looked into frequencies of words immediately following ‘so’
- You pickled the bigram conditional frequency distributions, and unpickled them to use in “BigramSpeak.py”

🡐 What was the point of this homework?

Basic corpus stats

The Bible

```
Word token count:    946,812
Word type count:      17,188
```

The Bible is over 2x as large.

Jane Austen novels

```
Word token count:    431,079
Word type count:      11,642
```

*On NLTK 3.7, you get 431,070 tokens and 11,645 types.

Top bigram frequencies

The Bible

```
, and 24944
of the 11541
the lord 7016
and the 6265
in the 5030
; and 3216
: and 3029
, that 2991
and he 2790
, the 2463
shall be 2461
to the 2152
all the 2138
and they 2086
him , 2037
unto the 2032
i will 1915
, which 1793
lord , 1709
of israel 1695
```

Jane Austen novels

```
, and 4748
. '' 2259
; and 1945
'' `` 1815
to be 1419
of the 1414
, '' 1393
in the 1125
, i 1117
. i 1069
. `` 984
it was 935
. she 895
; but 886
, that 815
, as 773
, she 759
she had 743
i am 741
she was 701
```

What do you notice?

Top 20 *so*-initial bigrams

The Bible

```
so that 192        so did 29
so the 136         so david 29
so shall 109       so be 22
so they 85         so great 16
so he 73           so when 15
so , 68            so then 15
so is 48           so with 14
so will 44         so to 14
so it 39
so i 35
so much 33
so . 31
```

Predominantly used as conjunctive adv

Jane Austen novels

```
so much 206        so ; 19
so very 113        so ? 17
so , 78            so often 16
so well 61         so it 16
so many 56         so you 16
so long 50         so kind 15
so far 49          so great 14
so little 44       so entirely 11
so . 37
so i 36
so soon 23
so good 20
```

Predominantly used as adj/adv modifier ('intensifier')

Given $w1$, calculating probability of $w2$

After *so* (*w1*), what are the probabilities of the next word (*w2*) being *much*? How about *will*?

The Bible

- There are 1689 total "so ..." bigrams.
- Of them, 33 are "so much". Therefore, *much* has 33/1689*100 = 1.95% chance of being the next word.
- Of them, 44 are "so will". Therefore, *will* has 44/1689*100 = 2.60% chance of being the next word.

Jane Austen novels

- There are 1969 total "so ..." bigrams.
- Of them, 206 are "so much". Therefore, *much* has 206/1969 *100 = 10.46% chance of being the next word.
- Of them, only 1 is "so will". Therefore, *will* has 1/1969 *100 = 0.05% chance of being the next word.

Letting bigrams speak

"Bible Speak"

so i say they said jesus had not be, but when i was there was an house for thou hast spoken it to me; but they said in a man, he hath said jesus. for his when they shall i say unto thee? saith thus unto her; the king david. these cities. selah: it to his father which was come upon thy seed for his hand upon thee with

"Jane Austen Speak"

she had seen the house was the room for the same room - he might be more in my dear mrs smith; she was the world! but, i can. i am glad to have made a woman - it was so very happy with you may guess her own, and her to be no one can you are so. that you must be no longer than a most fortunate chance

Bigram Speak as a language model

- Is "Bible Speak" a language model?
  - Yes. It is a <u>bigram model</u> of the English language of the bible.
- Is "Bible Speak" a *good* language model?
  - Pretty decent, compared with:

Randomly picked from Bible word list	tabernacles stare eaters eliphalet sorcery admah cherish emptiers whoever undertake profiting canaanitess lips torches pleiades mahanaim eshban inclineth riblah prophets attend shelemiah treasurer plantation huntest shutting alush arisai
Unigram model: word frequencies are reflected	he jeduthun he well did before the he among, all the that the wicked: because; day of of bring upon we was i by: feared of and: made noise a they with had of all tiberias of: when

  - How to make it better?

Bible, bigrams vs. trigrams

Bigram model

so i say they said jesus had not be, but when i was there was an house for thou hast spoken it to me; but they said in a man, he hath said jesus. for his when they shall i say unto thee? saith thus unto her; the king david. these cities. selah: it to his father which was come upon thy seed for his hand upon thee with

Trigram model

in the day of his own soul, and all their soul from going down to hell with him, and all his servants; how shall ye not read this letter in the house: therefore they called rebekah, jacob and israel. now ziba had fifteen sons and his sons, and the people: but i would that all they from their evil, that he may eat, and the king's

Austen, bigrams vs. trigrams

Bigram model

she had seen the house was the room for the same room - he might be more in my dear mrs smith; she was the world! but, i can. i am glad to have made a woman - it was so very happy with you may guess her own, and her to be no one can you are so. that you must be no longer than a most fortunate chance

Trigram model

it was to take a box for tuesday. " i do assure you. i shall never be a better match for my part to make his fortune, and that you will be very glad, " he replied; " i am quite of the two miss steeles to spend in bath; sir walter elliot: an extraordinary fate. the miss musgrove's, it will be very sure you must know

Bigram Speak vs. linguistic knowledge

- What kind of **linguistic knowledge** does the program have?
  - Phonetics? phonology? morphology? syntax? semantics? pragmatics?
  - Truth is, it does not have linguistic knowledge beyond:
    - Available words in a particular sublanguage
    - *Positive* proof of a word following another word, and its likelihood

← It showcases a purely **data-driven**, **statistical**, and **knowledge-poor** approach to language modeling.

← **ChatGPT** is essentially an n-gram language model too at its core, but a *much more* sophisticated one!

Estimating sentence probability

She was not afraid.

- How likely is this sentence in…
  - The Bible?
  - Jane Austen novels?

Sentence probability: TAKE 1

She was not afraid.

- In each corpus, find out **what proportion of all sentences** are exactly "She was not afraid."
  - Bible: 0/29812 → 0.00 probability
  - Austen: 0/15941 → 0.00 probability
- Is this a viable approach?
  - No. Natural language sentences are highly **productive**; the vast majority of human sentences are not repeated verbatim.

Sentence probability: TAKE 2

She was not afraid.

- Find the **probability of each word**, then **multiply**.

    → NEXT SLIDE

```
>>> sent = "she was not afraid .".split()
>>> sent
    ['she', 'was', 'not', 'afraid', '.']

>>> [b_tokfd.freq(x) for x in sent]
    [0.0010371647169659867, 0.004776027342281256, 0.007160872485773311,
    0.00020384194539148216, 0.027673920482630013]

>>> import numpy
>>> numpy.prod([b_tokfd.freq(x) for x in sent])
    2.0009891005865551e-13

>>> [a_tokfd.freq(x) for x in sent]
    [0.011819426079291066, 0.012977010694318789, 0.010657201846567843,
    0.00023894031131834736, 0.031289581738464755]
>>> numpy.prod([a_tokfd.freq(x) for x in sent])
    1.2220906621589035e-11
```

The sentence has a higher chance in Jane Austen novels.

But is this good enough?

Sentence probability: TAKE 2

She was not afraid.

- Find the **probability of each word**, then **multiply**.
  - P('She was not afraid.')

  = P('she') * P('was') * P('not') * P('afraid') * P('.')

- Problem?
  - "Was she not afraid." and even "Not she afraid was." will end up with the exact same probabilities. Sentences are more than just word salad…
  - This **unigram-based probability estimation** is still inadequate.

Sentence probability: TAKE 3

She was not afraid.

- We take **conditional probability of the bigrams** into consideration.
  - P('was'|'she'), P('not'| 'was'), …

    ← probability of 'was' following 'she', etc.
- So, we can multiply together:
  - P('was'|'she') * P('not'|'was') * P('afraid'|'not') * P('.'|'afraid')

  ← Anything missing?

  ← Yep: the probability of "She" being the first word, and "." being the last word of the sentence.

Sentence probability: TAKE 3

<s>She was not afraid.</s>

Pseudo tokens indicating beginning and end of sentence

- We take **conditional probability of the bigrams** into consideration.
- P('She was not afraid.') can be estimated as:
  - P('she'|<s>)❶ * P('was'|'she') * P('not'|'was') * P('afraid'|'not') * P('.'|'afraid') * P(</s>|'.')❷

← When processing bigrams in Homework #2, we did not take **sentence boundaries** into consideration.

← We will substitute ❶ with unigram probability P('she'), and just disregard ❷

```
>>> sent
['she', 'was', 'not', 'afraid', '.']

>>> b_probs = [b_tokfd.freq('she'), b_bigramcfd['she'].freq('was'),
    b_bigramcfd['was'].freq('not'), b_bigramcfd['not'].freq('afraid'),
    b_bigramcfd['afraid'].freq('.')]
>>> b_probs
[0.0010371647169659877, 0.06415478615071284, 0.03339230429013706,
0.005162241887905605, 0.16580310880829016]
>>> numpy.prod(b_probs)
1.9017534156537365e-09

>>> a_probs = [a_tokfd.freq('she'), a_bigramcfd['she'].freq('was'),
    a_bigramcfd['was'].freq('not'), a_bigramcfd['not'].freq('afraid'),
    a_bigramcfd['afraid'].freq('.')]
>>> a_probs
[0.011819426079291066, 0.13758586849852797, 0.06506971755452227,
0.0010883761427949495, 0.02912621359223301]
>>> numpy.prod(a_probs)
3.3543794097952598e-09
```

The sentence again has a higher chance in Jane Austen novels, with a lower margin this time

More on sentence probability estimation

- SLP ed.3, ch.3 N-gram Language Models
  - https://web.stanford.edu/~jurafsky/slp3/3.pdf#page=6
  - Bigram counts and probabilities with these words:
    - *I, want, to, eat, Chinese, English, food, lunch, spend, …*
  - How to estimate sentence probability of:
    - <s> I want English food </s>

$$
\begin{aligned}
P(\texttt{<s> i want english food </s>}) &= P(\texttt{i}|\texttt{<s>})P(\texttt{want}|\texttt{i})P(\texttt{english}|\texttt{want}) \\
&\qquad P(\texttt{food}|\texttt{english})P(\texttt{</s>}|\texttt{food}) \\
&= .25 \times .33 \times .0011 \times 0.5 \times 0.68 \\
&= .000031
\end{aligned}
$$

General, LARGER n-gram stats

- The Bible and Austen bigram stats reflect their unique topical content and linguistic traits.
- Can we find n-gram stats that are extracted from…
  - more GENERAL-domain text?
  - LARGER amounts of text?

Data Resources:

- NLTK Corpora Index [page]
- Natural Language Corpus Data by Peter Norvig [link]
- Google Books Ngram Viewer Data [link] *(Slow? Try FireFox.)*
- COCA *n*-gram lists at BYU [link]

n-grams and statistical NLP

- It is possible to obtain a highly detailed & accurate set of *n*-gram statistics.
  - How? Through **corpus data**.

- **Corpus-sourced, large-scale *n*-grams** are one of the biggest contributors to the recent advancement of <u>statistical</u> natural language processing (NLP) technologies.

- Used for: spelling correction, machine translation, speech recognition, information extraction...

  → JUST ABOUT ANY NLP APPLICATION

Norvig's data: 1- & 2-grams

- count_1w.txt

```
the     23135851162
of      13151942776
and     12997637966
to      12136980858
a       9081174698
in      8469404971
for     5933321709
is      4705743816
on      3750423199
that    3400031103
by      3350048871
this    3228469771
with    3183110675
i       3086225277
you     2996181025
it      2813163874
not     2633487141
or      2590739907
be      2398724162
are     2393614870
from    2275595356
at      2272272772
as      2247431740
your    2062066547
```

- count_2w.txt

```
you graduate    117698
you grant       103633
you great       450637
you grep        120367
you grew        102321
you grow        398329
you guess       186565
you guessed     295086
you guys        5968988
you had 7305583
you hand        120379
you handle      336799
you hang        144949
you happen      627632
you happy       603963
you has 198447
you hate        637001
you have        135266690
you havent      134438
you having      344344
you he  199259
you head        205910
you hear        2963179
you heard       1267423
```

Where do they come from?

Big data at our fingertips

- How to process data resources, downloaded from the internet?
  - From Norvig's data page https://norvig.com/ngrams/, download:
    - Word 1-grams: `count_1w.txt`

Huge file. Wait until your browser fully loads the page before hitting "save as"!

- Data derived from the [Google Web Trillion Word Corpus]()
  - Essentially unigram frequency data
  - Top 333K entries, taken from Google's original data (which is much bigger)

← Let's process this file into a Python data object.

← How to do this?

Step 1: stare at the file.

norvig.com/ngrams/count_1w.txt

the	23135851162
of	13151942776
and	12997637966
to	12136980858
a	9081174698
in	8469404971
for	5933321709
is	4705743816
on	3750423199
that	3400031103
by	3350048871
this	3228469771
with	3183110675

Sorted by frequency

One word per line, followed by count

Separated by white space: most likely a TAB

Step 2: read in as list of lines

```
>>> f = open('count_1w.txt')
>>> lines = f.readlines()
>>> f.close()
>>> lines[0]
    'the\t23135851162\n'
>>> lines[1]
    'of\t13151942776\n'
>>> len(lines)
    333333
>>>
```

May also need: `encoding='utf-8'`

Because of the "one entry per line" format of the original file, `.readlines()` is better suited.

Step 3: decide on data structure.

```
>>> goog1w_rank[:5]
[('the', 23135851162), ('of', 13151942776), ('and', 12997637966),
('to', 12136980858), ('a', 9081174698)]
>>> goog1w_rank[0]
('the', 23135851162)
>>> goog1w_rank[-1]
('golgw', 12711)


>>> goog1w_fd['platypus']
565585
>>> goog1w_fd.most_common(5)
[('the', 23135851162), ('of', 13151942776), ('and', 12997637966),
('to', 12136980858), ('a', 9081174698)]
>>> type(goog1w_fd)
<class 'nltk.probability.FreqDist'>
```

(1) **a list** where each item is **(word, count)** tuple.

We will keep the original **order**, which reflects the **frequency rank**.

(2) **a frequency distribution**

nltk.FreqDist where each word is mapped to its count

Step 4: experiment with a small copy.

```
>>> mini = lines[:5]
>>> mini
    ['the\t23135851162\n', 'of\t13151942776\n', 'and\t12997637966\n',
    'to\t1213698085 8\n', 'a\t9081174698\n']
>>> mini[0].split()
    ['the', '23135851162']
>>> for line in mini:
...     (word, count) = line.split()
...     tu = (word, int(count))
...     print(tu)
...
    ('the', 23135851162)
    ('of', 13151942776)
    ('and', 12997637966)
    ('to', 12136980858)
    ('a', 9081174698)
>>>
```

Mini version of `lines`

Build
(word, count) tuple
from each line

To be continued… in shell

- Demonstration in IDLE shell
- Make sure to check the posted shell session!

- Last step: pickle both data

```
>>> import pickle
>>> f = open('goog1w_rank.pkl', 'wb')
>>> pickle.dump(goog1w_rank, f, -1)
>>> f.close()
>>> 
>>> f2 = open('goog1w_fd.pkl', 'wb')
>>> pickle.dump(goog1w_fd, f2, -1)
>>> f2.close()
>>> 
```

Wrap-up

- Exercise #5 out
  - Process Norvig's bigram data
- HW #1 grades are in
  - Check ANSWER KEY, feedback

- Next class (Thu):
  - More on big-data n-gram stats
  - Processing a corpus
  - NLTK's built-in corpus methods