Language Technology - EDAN20
Assignment 2 - Fall 2020

Language Models
n-gram statistics - Probability of a Sentence

Hicham Mohamad (hi8826mo-s)

November 1, 2020

1 Introduction

In this second assignment, we will write Python programs to find n-gram statistics i a corpus of approximately one million words and to determine the probability of a sentence. After collecting the corpus, we will deal with building a segmenter and discussing the perplexity concept using different language models. At the end, as an application of n-grams, we will execute the Jupyter notebook by Peter Norvig [5].

2 Collecting the corpus

Before we can work with words, we need first some text, possibly from a file. Then we can break the text into words. As in assignment 1, we retrieve the corpus of novels by Selma Lagerlöf from this URL: `https://github.com/pnugues/ilppp/blob/master/programs/corpus/Selma.txt` . The text of these novels was extracted from Lagerlöf arkivet at Litteraturbanken. After collecting Selma we could analyse it through different tasks:

1. We run the **concordance program** to print the lines containing a specific word, for instance *Nils* with width equal to 25.
2. We run the **tokenization program** on the corpus.
3. We count the number of **unique words**, or unigrams, in the original corpus, by implementing the function `count_unigrams()` and then we put them in a dictionary. It results that from a vocabulary of 923485 words we have 44256 unique words.
4. We set all the words in **lowercase**. Here is the first 10: ['selma', 'lagerlöf', 'nils', 'holgerssons', 'underbara', 'resa', 'genom', 'sverige', 'första', 'bandet']

3 Segmenting a corpus

In this task, we need to write a program to tokenize the text, insert `<s>` and `</s>` tags to delimit sentences, and set all the words in lowercase letters. At the end, we will create a list of the words. Here we can illustrate the principal steps:

1. **Detecting sentence boundaries**: Here we write a regex called `sentence_boundaries` that matches a punctuation, a sequence of spaces, and an uppercase letter. In the regex, we need to remember the value of the uppercase letter using a backreference. Using the Unicode regexes for the letters and the spaces, we get:

```
sentence_boundaries = r'[.;:?!]+\p{Z}+(\p{Lu})'
```

2. **Sentence boundary markup**: we write a regex `sentence_markup` to insert `<s>` and `</s>` tags to delimit sentences, where the first letter of the sentence is in a regex backreference:

```
sentence_markup = r' </s>\n<s> \1'
```

3. **Normalizing**: the result should be a normalized text without punctuation signs. At the end, we create a file with one sentence per line where all the sentences are delimited with `<s>` and `</s>` tags.

Here we have the resulting function *segment_sentences()*

```
def segment_sentences(text):
    # matches a punctuation, a sequence of spaces, and an uppercase letter
    sentence_boundaries = r'[.;:?!]\p{Z}+(\p{Lu})'

    # sentence boundary markup
    sentence_markup = r' </s>\n<s> \1'

    # Substitution: replace the matched boundaries with the sentence boundary markup
    text = re.sub(sentence_boundaries, sentence_markup, text)

    # Insert markup codes in the beginning and end of the text
    text = '<s> ' + text + ' </s>'

    # Replace the space duplicates with one space
    # Z refers to seperators
    text = re.sub(r'\p{Z}+', r' ', text)

    # remove the punctuation signs
    text = re.sub(r'[.;:?!]', r'', text)
    #text = re.sub(r'^', r'<s> ', text)
    #text = re.sub(r'$', r' </s>', text)

    text = text.lower()
    return text
```

4 Counting unigrams and bigrams

After the word segmentation task, we create a list of words from our strings. Now, we need to count the word and store the unigram and bigram counts/frequencies. By running the provided programs we can compute the frequency of unigrams and bigrams of the training set.

The bigram model approximates the probability of a word given all the previous words by using only the **conditional probability** of the preceding word. When we use a bigram model to predict the conditional probability of the next word, we are thus making the following approximation:

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-1})$$

we can compute the probability of a complete **word sequence** by

$$P(w_1^n) \approx \prod_{k=1}^{n} P(w_k|w_{k-1})$$

Consequently, if the probability of any word in the set is zero, the entire probability of the set is zero.

4.1 Potential number and real number

Given a vocabulary of size N, the **possible number** of bigrams is N^2, and with 4-grams, it amounts to N^4. But the problem is that no corpus yet has the size to cover the corresponding word combinations, and instead we have the so called **real number**.

4.2 Data sparsity

The n-gram model, like many statistical models, is dependent on the training set/corpus. Because any corpus is limited, some perfectly accepted english word sequences are bound to be missing from it. That is, we will have many cases of **putative zero probability n-grams** that should really have some non-zero probability. This phenomenon is referred to as **data sparsity**, where any particular sequence that occurs in the test set may simply never have occured in the training set, i.e. these zeros do occur in the test set and not in the training set.

4.3 Smoothing

We need to use smoothing techniques to keep a language model from assigning zero probability to words that appear in a test set in an **unseen context**, for example they appear after a word they never appeared after in training. These techniques will shave off a bit of probability mass from some more frequent events and give it to the events we have never seen. There are a variety of ways to do smoothing like Laplace smoothing, Good-Turing, Back-off, and Kneser-Ney smoothing.

5 Perplexity - Computing the likelihood of a sentence

In practice we don't use **raw probability** as our metric for evaluating language models, but a variant called perplexity. The perplexity (sometimes called PP for short) of a language model on a test set is the **inverse probability** of the test set, normalized by the number of words. For a test set $W = w_1 w_2 \cdots w_N$, if we are computing the perplexity of W with a bigram language model, we get:

$$PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}}$$

In this task, we need to write a program to compute the sentence's probability using unigrams, and another one to compute it using bigrams. with the sentence *Det var en gång en katt som hette Nils*, the results are tabulated below.

Unigram model

wi	C(wi)	#words	P(wi)
det	21108	1041560	0.020265755213333847
var	12090	1041560	0.011607588617074388
en	13514	1041560	0.01297476861630631
gång	1332	1041560	0.0012788509543337724
en	13514	1041560	0.01297476861630631
katt	16	1041560	1.5361573025077767e−05
som	16288	1041560	0.015638081339529167
hette	97	1041560	9.312953646453396e−05
nils	87	1041560	8.352855332386037e−05
</s>	59047	1041560	0.056690925150735434

Prob. unigrams: 5.3651155337425844e−27
Geometric mean prob.: 0.0023602494649885993
Entropy rate: 8.726844932328587
Perplexity: 423.68402782577465

Bigram model

wi	wi+1	Ci, i+1	C(i)	P(wi+1|wi)
<s>	det	5672	59047	0.09605907158704083
det	var	3839	21108	0.1818741709304529
var	en	712	12090	0.058891645988420185
en	gång	706	13514	0.052242119283705785

gång	en	20	1332	0.015015015015015015
en	katt	6	13514	0.0004439840165754033
katt	som	2	16	0.125
som	hette	45	16288	0.002762770137524558
hette	nils	0	97	0.0 *backoff: 8.352855332386037e-05
nils	</s>	2	87	0.022988505747126436

Prob. bigrams: 2.376169768780815e-19
Geometric mean prob.: 0.013727382866049192
Entropy rate: 6.186799588766882
Perplexity: 72.84709764111103

5.1 My test set results

In this task, we select five sentences as a test set and run the implemented `unigram_lm()` and `bigrams_lm()` programs on them. The obtained results are tabulated below.

5.1.1 Sentence 1

sentence 1 = 'lite kunskap är en farlig sak '

Unigram model

wi	C(wi)	#words	P(wi)
lite	45	1041560	4.320442413303122e-05
kunskap	14	1041560	1.3441376396943047e-05
är	6290	1041560	0.006039018395483697
en	13514	1041560	0.01297476861630631
farlig	40	1041560	3.840393256269442e-05
sak	205	1041560	0.0001968201543838089
</s>	59047	1041560	0.056690925150735434

Prob. unigrams: 1.9498300024612508e-23
Geometric mean prob.: 0.0005697886857557694
Entropy rate: 10.777285405072845
Perplexity: 1755.0366039185164

Bigram model

wi	wi+1	Ci,i+1	C(i)	P(wi+1|wi)
<s>	lite	0	59047	0.0 *backoff: 4.320442413303122e-05
lite	kunskap	0	45	0.0 *backoff: 1.3441376396943047e-05
kunskap	är	0	14	0.0 *backoff: 0.006039018395483697
är	en	304	6290	0.04833068362480127
en	farlig	6	13514	0.0004439840165754033
farlig	sak	0	40	0.0 *backoff: 0.0001968201543838089
sak	</s>	34	205	0.16585365853658537

Prob. bigrams: 2.4565364591697616e-21
Geometric mean prob.: 0.0011369999855527747
Entropy rate: 9.78055204876345
Perplexity: 879.5074869889572

5.1.2 Sentence 2

sentence2 = ' tidigt till sängs och tidigt att stiga gör en man frisk '

Unigram model

wi	C(wi)	#words	P(wi)
tidigt	38	1041560	3.64837359345597e−05
till	9139	1041560	0.008774338492261608
sängs	18	1041560	1.728176965321249e−05
och	36356	1041560	0.03490533430623296
tidigt	38	1041560	3.64837359345597e−05
att	28020	1041560	0.02690195476016744
stiga	90	1041560	8.640884826606244e−05
gör	355	1041560	0.000340834901493913
en	13514	1041560	0.01297476861630631
man	2322	1041560	0.002229348285264411
frisk	69	1041560	6.624678367064787e−05
</s>	59047	1041560	0.056690925150735434

Prob. unigrams: 6.063662305241324e−37
Geometric mean prob.: 0.0009591677848954893
Entropy rate: 10.025929175084222
Perplexity: 1042.570461339003

Bigram model

wi	wi+1	Ci, i+1	C(i)	P(wi+1|wi)
<s>	tidigt	3	59047	5.080698426677054e−05
tidigt	till	1	38	0.02631578947368421
till	sängs	18	9139	0.001969580916949338
sängs	och	1	18	0.05555555555555555
och	tidigt	1	36356	2.750577621300473e−05
tidigt	att	2	38	0.05263157894736842
att	stiga	28	28020	0.0009992862241256246
stiga	gör	0	90	0.0 *backoff: 0.000340834901493913
gör	en	6	355	0.016901408450704224
en	man	117	13514	0.008657688323220364
man	frisk	0	2322	0.0 *backoff: 6.624678367064787e−05
frisk	</s>	10	69	0.14492753623188406

Prob. bigrams: 1.0134118069871207e−31
Geometric mean prob.: 0.002613056679059623
Entropy rate: 8.580045866582262
Perplexity: 382.69357416306656

5.1.3 Sentence 3

sentence3 = 'ju större de är desto hårdare faller de '

Unigram model

wi	C(wi)	#words	P(wi)
ju	1250	1041560	0.0012001228925842006
större	150	1041560	0.00014401474711010407
de	11942	1041560	0.011465494066592419
är	6290	1041560	0.006039018395483697
desto	44	1041560	4.224432581896386e−05
hårdare	8	1041560	7.680786512538884e−06
faller	47	1041560	4.5124620761165944e−05
de	11942	1041560	0.011465494066592419

```
</s>      59047    1041560          0.05669092515073543 4
=====================================================
Prob. unigrams: 1.1389004736737127e-28
Geometric mean prob.: 0.0007855341786154558
Entropy rate: 10.314038330960237
Perplexity: 1273.019083348546

Bigram model
=====================================================
wi        wi+1       Ci,i+1       C(i)        P(wi+1|wi)
=====================================================
<s>       ju         21         59047      0.00035564888986739377
ju        större     0          1250       0.0 *backoff:  0.00014401474711010407
större    de         0          150        0.0 *backoff:  0.011465494066592419
de        är         59         11942      0.004940545972198961
är        desto      0          6290       0.0 *backoff:  4.224432581896386e-05
desto     hårdare    0          44         0.0 *backoff:  7.680786512538884e-06
hårdare   faller     0          8          0.0 *backoff:  4.5124620761165944e-05
faller    de         0          47         0.0 *backoff:  0.011465494066592419
de        </s>       102        11942      0.008541282867191425
=====================================================
Prob. bigrams: 4.160060663585748e-30
Geometric mean prob.: 0.0005438204290664426
Entropy rate: 10.844582031130408
Perplexity: 1838.842284238317
```

5.1.4 Sentence 4

sentence 4 = 'hur vacker skulle världen vara om det fanns en regel för att gå runt i labyrinter'

```
Unigram model
=====================================================
wi        C(wi)       #words        P(wi)
=====================================================
hur         1996      1041560          0.0019163562348784515
vacker      209       1041560          0.00020066054764007835
skulle      5433      1041560          0.005216214140327 97
världen     363       1041560          0.00034851568800645187
vara        1803      1041560          0.0017310572602634 51
om          8075      1041560          0.007752793886093936
det         21108     1041560          0.020265755213333847
fanns       702       1041560          0.0006739890164752871
en          13514     1041560          0.012974768616306 31
regel       2         1041560          1.920196628134721e-06
för         9443      1041560          0.009066208379738086
att         28020     1041560          0.0269019547601674 4
gå          1590      1041560          0.0015265563193671032
runt        154       1041560          0.0001478551403663735
i           16508     1041560          0.015849302968623986
labyrinter 1          1041560          9.600983140673605e-07
</s>        59047     1041560          0.05669092515073543 4
=====================================================
Prob. unigrams: 1.5161591885734908e-49
Geometric mean prob.: 0.001343628187008172
Entropy rate: 9.539650318399216
Perplexity: 744.2535142305092

Bigram model
=====================================================
wi        wi+1       Ci,i+1       C(i)        P(wi+1|wi)
```

<s>	hur	238	59047	0.004030687418497129
hur	vacker	3	1996	0.001503006012024048
vacker	skulle	0	209	0.0 *backoff: 0.00521621414032797
skulle	världen	0	5433	0.0 *backoff: 0.00034851568800645187
världen	vara	0	363	0.0 *backoff: 0.001731057260263451
vara	om	5	1803	0.0027731558513588465
om	det	558	8075	0.06910216718266254
det	fanns	253	21108	0.011985976880803486
fanns	en	74	702	0.10541310541310542
en	regel	1	13514	7.399733609590054e−05
regel	för	0	2	0.0 *backoff: 0.009066208379738086
för	att	2932	9443	0.3104945462247167
att	gå	360	28020	0.01284796573875803
gå	runt	3	1590	0.0018867924528301887
runt	i	13	154	0.08441558441558442
i	labyrinter	0	16508	0.0 *backoff: 9.600983140673605e−07
labyrinter	</s>	1	1	1.0

Prob. bigrams: 1.88910284270759e−39
Geometric mean prob.: 0.005273909614054819
Entropy rate: 7.566911438615527
Perplexity: 189.61265421292558

5.1.5 Sentence 5

sentence 5 = 'se inget ont hör inget ont tala inget ont'

Unigram model

wi	C(wi)	#words	P(wi)
se	1989	1041560	0.001909635546679998
inget	34	1041560	3.264334267829026e−05
ont	150	1041560	0.00014401474711010407
hör	222	1041560	0.00021314182572295404
inget	34	1041560	3.264334267829026e−05
ont	150	1041560	0.00014401474711010407
tala	845	1041560	0.0008112830753869196
inget	34	1041560	3.264334267829026e−05
ont	150	1041560	0.00014401474711010407
</s>	59047	1041560	0.056690925150735434

Prob. unigrams: 1.9449565461801393e−36
Geometric mean prob.: 0.00026846704976591837
Entropy rate: 11.862967349276994
Perplexity: 3724.85189848035

Bigram model

wi	wi+1	Ci, i+1	C(i)	P(wi+1|wi)
<s>	se	196	59047	0.003319389638762342
se	inget	0	1989	0.0 *backoff: 3.264334267829026e−05
inget	ont	6	34	0.17647058823529413
ont	hör	0	150	0.0 *backoff: 0.00021314182572295404
hör	inget	0	222	0.0 *backoff: 3.264334267829026e−05
inget	ont	6	34	0.17647058823529413
ont	tala	0	150	0.0 *backoff: 0.0008112830753869196

```
tala     inget    0     845    0.0 *backoff:   3.264334267829026e-05
inget    ont      6     34     0.17647058823529413
ont      </s>     23    150    0.15333333333333332
=========================================================
Prob. bigrams: 1.682429136522422e-26
Geometric mean prob.: 0.002646023385115938
Entropy rate: 8.561958472689598
Perplexity: 377.92560928413127
```

6 Reading

As an application of n-grams, we need to execute the Jupyter notebook by Peter Norvig here. After running all the cells and understand the code, we experiment with the following long string:

"*How beautiful would be the world if there were a rule to turn into labyrinths.*"

First we remove all the punctuation and white spaces from this string, and then we set it in lower-case letters. By adding a cell at the end of Section 7 in Norvig's notebook, we use our string and run the notebook cell with the `segment()` and `segment2()` functions. In the following, we get the obtained results:

```
my_text = Howbeautifulwouldbetheworldiftherewerearuletoturnintolabyrinths

print(segment(my_text))
[Howbeautifulwouldbe, theworldiftherewerea, rule, toturnintolabyrinths]

print(segment2(my_text))
[Howbeautifulwouldbe, the, world, if, there, were, a, rule, toturnintolabyrinths]
```

6.1 Conclusion

N-grams do a better and better job of modeling the training corpus as we increase the value of N. The output confirms that the bigram model segmenter is better than the unigram one, but hundreds of billions of words still not enough to get good result. Thus, trillions of words could give more efficient segmentation result.

7 Appendix

References

[1] Pierre Nugues, *Language processing with Perl and Prolog*, 2nd edition, 2014, Springer.

[2] P. Nugues, Language Technology - EDAN20, Lectures notes: `https://cs.lth.se/edan20/`

[3] D. Jurafsky, J. Martin, *Speech and Language Processing*, 3rd edition. Online: `https://web.stanford.edu/~jurafsky/slp3/`

[4] Pierre Nugues, Assignment 2: `https://github.com/pnugues/edan20/blob/master/notebooks/2-language_models.ipynb`

[5] Peter Norvig, *How To Do Things With Words and Counters*, jupyter notebook: `http://nbviewer.jupyter.org/url/norvig.com/ipython/How%20to%20Do%20Things%20with%20Words.ipynb`

[6] VanderPlas, A Whirlwind Tour of Python. O'Reilly 2016. Online reading: `https://jakevdp.github.io/WhirlwindTourOfPython/`

[7] Geron, Jupyter notebook on linear algebra from Machine Learning and Deep Learning in Python using Scikit-Learn, Keras and TensorFlow: `https://github.com/ageron/handson-ml2`