EDAN20 Language Technology
-Lab 2.

Nils Romanus

September 2022

1 Creating a Language Model

Using the regex library coupled with basic python operations for arithmetics and text manipulation one may create a language model based upon the Selma Lagerlöf corpus.

1.1 Segmenting a corpus

By using the regex pattern `r'[^(\p{L}|\p{Z}|,\.!?)]'` with the `sub` function from the regex module the text can be cleaned from non letter characters that are not white space or punctuations. Sentence boundaries can subsequently be detected by matching on the regex pattern `r'\p{P}\p{z}*(\p{Lu})'`. To mark the sentence boundaries one may substitute a boundary with the sentence markup `r' </s>\n<s> \1'`. Punctuations can be removed by using the pattern `r'\p{Z}{2,}|[^[/|\P{P}]]'` where the last block represents all punctuations except the `/` character in order to preserve the sentence markup. The final `segment_sentences` functions can subsequently be constructed by combining the aforementioned patterns - firstly marking sentence boundaries, secondly removing punctuations, and finally manually adding sentence markup to the beginning and end of the whole of the text.

1.2 Computing the likelihood of a sentence

By using uni- and bigram frequencies, generated from the `unigrams` and `bigrams` functions from the programs folder, one may compute the likelihood of a sentence. Using unigrams this can be done by iterating over all words in a sentence and estimating the probability for each word by computing its relative frequency in the corpus. The total probability is subsequently computed as the product of the probabilities for each word. The total probability can subsequently be standardized by computing its geometric mean. The entropy rate is computed by computing the negative base 2 logarithm of the total probability normalized by the number of words in the sentence. Finally, the perplexity metric is computed by raising 2 to the power of the entropy rate. A report containing these metrics coupled with the estimated probabilities for each word, using unigrams, for the sentence 'det var en gång en katt som hette nils', is displayed in table 1.

wi	C(wi)	#words	P(wi)
det	21108	1042133	0.020254612415114
var	12090	1042133	0.011601206371931414
en	13514	1042133	0.012967634649320192
gång	1332	1042133	0.0012781477987934362
en	13514	1042133	0.012967634649320192
katt	16	1042133	1.5353126712233466e-05
som	16288	1042133	0.01562948299305367
hette	97	1042133	9.30783306929154e-05
nils	87	1042133	8.348262649776947e-05
</s>	59326	1042133	0.05692747470812267

Prob. unigrams: 5.360900589783176e-27
Geometric mean prob.: 0.0023600639734006846
Entropy rate: 8.726958317932167
Perprelixty: 423.71732769560083

Table 1: Report using unigrams for the sentence 'det var en gång en katt som hette nils'

One may take a similar approach using bigrams. The probabilities of each possible bigram of a sentence can be computed by iterating over all bigrams that can be constructed from the sentence. The probability of each bigram, i.e the conditional probability of word w_{i+1} given w_i is given by the frequency of the bigram in the corpus divided by the frequency of the first word, in the bigram, in the corpus. The probability of the sentence is given by the product of the conditional probabilities scaled by the probability of the first word. The report for the sentence 'det var en gång en katt som hette nils', using bigrams, is displayed in table 2.

```
=====================================================
wi   wi+1   Ci,i+1   C(i)   P(wi+1|wi)
=====================================================
det      var      3839      21108      1042133      0.1818741709304529
var      en      712      12090      1042133      0.058891645988420185
en      gång      706      13514      1042133      0.052242119283705785
gång      en      20      1332      1042133      0.015015015015015015
en      katt      6      13514      1042133      0.0004439840165754033
katt      som      2      16      1042133      0.125
som      hette      45      16288      1042133      0.002762770137524558
hette      nils      0      97      1042133      0.0
*backoff 8.348262649776947e-05
nils      </s>      2      87      1042133      0.022988505747126436
=====================================================
Prob. unigrams: 2.7532974741265584e-19
Geometric mean prob.: 0.0139310969190684
Entropy rate: 6.165547331161066
Perprelixty: 71.78185650486967
```

Table 2: Report using bigrams for the sentence 'det var en gång en katt som hette nils'

In addition to the first test sentence one may also apply this function to a couple of test sentences. The results of the test sentences: 'Hej jag heter Nils', 'Idag är det tisdag' and 'hundar kan vara farliga' are shown in table 3.

```
=====================================================
wi   wi+1   Ci,i+1   C(i)   P(wi+1|wi)
=====================================================
hej      jag      0      3      1042133      0.0
*backoff 0.009126474260003282
jag      heter      4      9511      1042133      0.00042056566081379455
heter      nils      1      78      1042133      0.01282051282051282
=====================================================
Prob. unigrams: 1.552162952232786e-11
Geometric mean prob.: 0.0019848803317696223
Entropy rate: 8.976732254607093
Perprelixty: 503.80871027546954
=====================================================
wi   wi+1   Ci,i+1   C(i)   P(wi+1|wi)
=====================================================
idag      är      0      1      1042133      0.0
*backoff 0.006035697938746782
är      det      688      6290      1042133      0.10937996820349762
det      tisdag      0      21108      1042133      0.0
*backoff 5.7574225170875506e-06
=====================================================
Prob. unigrams: 6.042862972894413e-13
Geometric mean prob.: 0.0008816793852883758
Entropy rate: 10.147458251368949
Perprelixty: 1134.1991393764124
=====================================================
wi   wi+1   Ci,i+1   C(i)   P(wi+1|wi)
=====================================================
hundar      kan      0      24      1042133      0.0
*backoff 0.0019939873317513217
kan      vara      86      2078      1042133      0.04138594802694899
vara      farliga      0      1803      1042133      0.0
*backoff 3.7423246361069075e-05
=====================================================
Prob. unigrams: 3.566830403947855e-11
Geometric mean prob.: 0.0024438278795107732
Entropy rate: 8.676641605937776
Perprelixty: 409.1941205778324
```

Table 3: Report using bigrams for the test sentences

1.3 Online Prediction of Words

The possible word candidates from a piece of starting text can be generated using the bigram consisting of the piece of text and the string indicating the

start of a sentence. By iterating over all words in the corpus that starts with the letters in the starting text and computing the frequency of the afforementioned bigram one may generate the most probable candidates by returning the last word of the most frequently occuring bigram. These candidates are then sorted deterministically primarily on the frequency and secondarily alphabetically. Similarly, the most probable next word of a string can be computed using trigrams generated from the `trigram` function from the programs folder. This is done by iterating over all words in the corpus and computing the probability using the last 2 words combined with the word in the iteration as a trigram. If the trigram is not in the set of trigrams generated from the corpus, the probability is computed using the last word and the word in the iteration as a bigram. These candidates are then sorted deterministically primarily on the probability and secondarily alphabetically. These two functions can subsequently be combined to find the most probable word in the end of a sentence that starts with a given string.

2 Norvig's Statistical Natural Language Processing in Python

Norvig's segmenting program is based on the same principals as the one we used for computing the likelihood of a sentence and predicting words. Similarly, it uses uni- and bigrams generated in a large corpus and the relative frequencies of these uni- and bigrams. I.e having a language model learn which words, in which context, are probable. Our *segmenting* program in the initial part of the lab is completely rule-based as opposed to Norvig's segmenter. However, our predicting program is based upon the same principles as Norvig's. Figure 1 displays the result using firstly the unigram-based segmenter, and secondly the bigram segmenter. As one may note from figure 1 the results are identical and no conclusion, as to which method is superior, can therefore be reached. In order to compare the segmenting models one would need to evaluate them seperately using metrics as opposed to only looking at one example.

```
my_test = "hellothisisatestsentenceonwhichiwilltestthesegmenter"
print(segment(my_test))
print(segment2(my_test))
```

```
['hello', 'this', 'is', 'a', 'test', 'sentence', 'on', 'which', 'i', 'will', 'test', 'the', 'segmenter']
['hello', 'this', 'is', 'a', 'test', 'sentence', 'on', 'which', 'i', 'will', 'test', 'the', 'segmenter']
```

Figure 1: Caption