Université Libre de Bruxelles

INFO-Y404

Natural Language Processing

Project : Language Model

Daoud Yemni

23 mars 2018

Table des matières

1 Language Model **2**
1.1 Description 2
1.2 Results 3
1.2.1 Excerpts 3
1.2.2 Random outputs 4
1.2.3 Perplexities 5

2 Discussion **6**

1 Language Model

1.1 Description

A Probabilistic language model is a mathematical model allowing to identify a language or variety of a language, spell correction, sentence translation, etc... The model learn a language from a training set. In our project, the choice of language model is the *N-gram*. The *N-gram* is an algorithm that compute the probability of a sequence of size N. The computation is to determine the probability of the word (or letter) at index N against to N-1 previous words/letters. Precisely, in our project, N is 3. For example, for the sentence "We have a sunny day", the probability (3-grams) of the word "day" is $P(\text{"}day\text{"}|\text{"}asunny\text{"})$.

The process is simple. The model will get the text from a training file and count all 3-grams existing in the text. We store the counting into a matrix of size 27^2x27. The size is 27 because we have the 26 letters of the alphabet plus the underscore. The rows correspond to every possible 2-grams and the columns correspond to each possible letters. The computation of the probability of each 3-gram is defined like that :

$$P(w_n|w_{n-2}w_{n-1}) = \frac{count(w_{n-2}w_{n-1}, w_n)}{count(w_{n-2}w_{n-1}}$$

We can remark one problem in this computation. If we don't occur a 3-gram, the probability will be null. This is a problem because we can occur the 3-gram in the test but not in the training. To reduce this problem, we need to modify the computation of the probability. We will apply the **Stupid Backoff** algorithm. The **Stupid Backoff** is a simplified version of the **Kneser-Ney** algorithm. The **Kneser-Ney** algorithm is simply to compute the probability from the n-gram to the 1-gram :

$$P(w_n|w_{n-2}w_{n-1}) = \lambda_1 P(w_n|w_{n-2}w_{n-1}) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n) \quad s.t. \quad \sum_i \lambda_i = 1$$

The **Stupid Backoff** algorithm is an iterative algorithm. Firstly, we will compute the classic probability. If the count has no occurrence, we reduce the level of the N-gram, we re-compute the probability and we multiply this

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

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

by a constant factor $\alpha = 0.4$. We reduce the level of the N-grams until the unigram.

Now, we have compute the probability of each 3-gram. Our language model has learn. Now, we use our model to a text. In our case, we apply 3 models to the test file, each one correspond to a variety of the english language : US - GB - AUS. We compute the perplexity for each model with each sentence. Lower the perplexity is, more the sentence correspond to the language. The computation of the perplexity is the inverse probability of the test set, normalized by the number of words :

$$PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_n|w_{n-2}w_{n-1})}}$$

where W is the sentence and N is the length of the sentence.

1.2 Results

1.2.1 Excerpts

Here the excerpt for British English and American English for the bigram "iz" :

iz	GB	US
a	0.19663	0.22263
b	0.00173	0.00052
c	0.00047	0.00077
d	0.00225	0.00125
e	0.61124	0.61157
f	0.00047	0.00077
g	0.00112	0.00164
h	0.00225	0.00052
i	0.05056	0.07560
j	0.00019	0.00019
k	0.00031	0.00087
l	0.00801	0.00558
m	0.00157	0.00156
n	0.00031	0.00052
o	0.04157	0.03285
p	0.00063	0.00077
q	0.00013	0.00019
r	0.00063	0.00106
s	0.00112	0.00106
t	0.00157	0.00052
u	0.01573	0.01303
v	0.00016	0.00135
w	0.00110	0.00067
x	0.00031	0.00058
y	0.00112	0.00962
z	0.04494	0.02346
_	0.03146	0.01721

1.2.2 Random outputs

We generate a random output string of size 200 for each variety of English. The generation is the Shannon Visualization method. Firstly, we choose a random trigram (<s>, w) according to its probability. After, we iterate by choosing a random bigram (w, x) according to its probability until we reach the desired size of the input.

The random output of AUSTRALIAN English is :

> "hat grrom comanamsyclearted on in rebsraces mal by bolocconces farrolemeas thism the exper asbrtaffirume stries buthearsimitioneen abs gont has hei houraper as s injou"

The random output of GB English is :

> "ball on runtiondmare its ce to casithating nubtle coo the mat a supe thes thern frous at ran st coutfroble by ittery beenoft ce regook so plve what andotbanywayn s"

The random output of US English is :

> "x apporpron or ducturoventiciecolecove witual as of fals il prow posunumsesed to gailif mataspers hil is me woutheyeel maloplack of ory of ing ther wout tery of re ra"

The random outputs are unreadable but there is some real words in the random sentence.

1.2.3 Perplexities

line	AU	GB	US	Best
0	6.209	6.442	6.348	AU
1	6.208	6.463	6.430	AU
2	7.423	7.682	7.508	AU
3	5.516	5.375	5.872	GB
4	6.164	6.013	5.994	US
5	5.775	5.775	5.788	AU/GB
6	5.523	5.254	5.197	US
7	6.195	6.126	6.094	US
8	5.471	5.248	5.235	US

The perplexities are good. The **Stupid Backoff** works well. My algorithm doesn't recognize properly the British English.

2 Discussion

Do you need to run the test set on all three language models or is the score from a single variety model sufficient ?
In our case, I don't think we can only run a single variety because as we can see in the results, the score is very close to each one. So to determine which variety the document is, we need to run all three language models.

Would a unigram or bigram language model work as well ? Explain why (not).
In my algorithm, we use the bigram and unigram for the **Stupid Backoff**. I don't think this will work better if we reduce the level of the initial n-gram because we lost the precision of the language/variety. I think the language model will be more generic and it won't be able to recognize properly the variety in a document.

Do the language models show anything about similarity of the varieties ? Why (not) ?
The language models show in their models some similarities. We saw some trigram has the same probability in all of varieties. This is normal because we manipulate the same language so there is some words or a structure used in the 3 varieties.

Can you think of a better way to make a language variety guesser ?