ANISH ADNANI

D17B/01

NLP
(1)

EXPERIMENT 5

Aim: Implement N-Gram model for the given text input.

Theory:

N-gram model.

From the Markov Assumption, we can formally define N-gram models where $k=n-1$ as the following

$$P(w_i / w_1 w_2 \ldots w_{i-1}) \approx P(w_i / w_{i-(n-1)} \ldots w_{(i-1)})$$

And the simplest versions of the above are defined as the unigram model ($K=1$) and the Bigram model ($K=2$)

Unigram model ($K=1$)

$$P(w_1 w_2 \ldots w_n) \approx \prod P(w_i)$$

Bigram model ($K=2$)

$$P(w_i / w_1 w_2 \ldots w_{i-1}) \approx P(w_i / w_{i-1})$$

These equations can be extended to compute trigrams, 4-grams, 5-grams, etc. In general, this is an insufficient model of language because sentences often have long distance dependencies. For example, the subject of a sentence may be at the start whilst our next word to be predicted occurs more than 10 words later.

Chain Rule

In general cases, the formula is as follows:

$$P(x_1, x_2, \ldots x_n) = P(x_1)\, P(x_2/x_1) \ldots P(x_n/x_1 \ldots x_{n-1})$$

The chain rule applied to compute the joined probability of words in a sequence is therefore

$$P(w_1 w_2 \ldots w_n) = \prod P(w_i / w_1 w_2 \ldots w_{i-1})$$

For example

P("its water is so transparent") = P(its) * P(water/its) * P(is/its water) * P(so/its water is) * P(transparent/its water is so)

Markov Assumption Property

A stochastic process has the markov property if the conditional probability distribution of future states of the process (conditional on both past and present states) depends only upon the present state, not on the sequence of events that preceded it. A process with this property is called markov process.

In other words, the probability of the next word can be estimated given only the previous K number of words

For example, if K=1:

P(transparent / its water is so) ≈ P(transparent / so)

or if $k=2$:
P(transparent | its water is so) ≈ P(transparent | is so)

General equation for the markov assumption, $k=i$

$$P(w_i | w_1 w_2 \ldots w_{i-1}) \approx P(w_i | w_{i-k} \ldots w_{i-1})$$

⇒ Challanges of N-gram model and techniques to overcome it:

1) Sensitivity to the training corpus
The N-gram model, like many statistical models, is significantly dependent on the training corpus. As a result, the probabilities often encode particular facts about a given training corpus. Besides, the performance of the N-gram model varies with the change in the values of N

Technique to overcome it:
We can have a language task in which we know all the words that can occur, and hence we know the vocabulary size V in advance. The closed vocabulary assumption assumes there are no unknown words, which is unlikely in practical scenarios.

2) Smoothing

A notable problem with the maximum likelihood estimation approach is sparse data. meaning, any N-gram that appeared a sufficient number of times might have a reasonable estimate for its probability. But because any corpus is limited, some perfectly acceptable english word sequences are bound to be missing from it.

Technique to overcome it

one way to solve this is to start with a fixed vocabulary and convert out of vocabulary words in training to UNK pseudo-word.

Conclusion:

Thus, we have successfully studied and implemented N-Gram model for the given text input

Code

```
import nltk
from nltk import word_tokenize
from nltk import bigrams, trigrams
test_list = ['Hi','How are you?','I am from VESIT', 'Nice to meet you']
count=0
new_list=[]
for i in test_list:
    if count==0:
        i="(eos) "+i+" (eos)"

    else:
        i=i+" (eos)"
    new_list.append(i)
    count+=1

word_list=[]
for i in new_list:
    x=i.split()
    word_list.extend(x)
res = [(x, i.split()[j + 1]) for i in new_list
for j, x in enumerate(i.split()) if j < len(i.split()) - 1]
text="Hello How are you? We are from VESIT Nice to meet you"
print("Orignal Text->",text)
unigrams = word_tokenize(text)
print("<--Unigrams-->")
print(unigrams)
bigrms=nltk.bigrams(text.split())
print ("<--Bigrams-->")
print(*map(' '.join, bigrms), sep=', ')
trigrms =nltk.trigrams(text.split())
print( "<--Trigrams-->")
print(*map(' '.join, trigrms), sep=', ')
print()
print("<--Before applying Smoothing Probability Table-->")

def prob_calc(a,b):
    prob=(res.count((b,a))+1)/(word_list.count(b)+len(set(word_list)))
    return prob
wordset=list(set(word_list))
print("\t",end="")
for i in wordset:
    print(" ",i, end="")
print()
```

```
for i in wordset:
    print(i,":\t", end="")

    for j in wordset:
        ans=prob_calc(j,i)
        print("%.3f "%ans,end="")

    print()

print()
print("<--After applying Smoothing Probability Table-->")
def prob_calc(a,b):
    prob=(res.count((b,a))+1)/(word_list.count(b)+len(set(word_list))+1)
    return prob
wordset=list(set(word_list))
print("\t",end="")

for i in wordset:
    print(" ",i, end="")

print()

for i in wordset:
    print(i,":\t", end="")
    for j in wordset:
        ans=prob_calc(j,i)
        print("%.3f "%ans,end="")

    print()
```

Output

jupyter anish-exp5-nlp Last Checkpoint: 14 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

```
Orignal Text-> Hello How are you? We are from VESIT Nice to meet you
<--Unigrams-->
['Hello', 'How', 'are', 'you', '?', 'We', 'are', 'from', 'VESIT', 'Nice', 'to', 'meet', 'you']
<--Bigrams-->
Hello How, How are, are you?, you? We, We are, are from, from VESIT, VESIT Nice, Nice to, to meet, meet you
<--Trigrams-->
Hello How are, How are you?, are you? We, you? We are, We are from, are from VESIT, from VESIT Nice, VESIT Nice to, Nice to mee
t, to meet you

<--Before applying Smoothing Probability Table-->
          to  you?  are  VESIT  I  Hi  am  you  from  (eos)  Nice  meet  How
to :    0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.143 0.071
you? :  0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.143 0.071 0.071 0.071
are :   0.071 0.143 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071
VESIT : 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.143 0.071 0.071 0.071
I :     0.071 0.071 0.071 0.071 0.071 0.071 0.143 0.071 0.071 0.071 0.071 0.071 0.071
Hi :    0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.143 0.071 0.071 0.071
am :    0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.143 0.071 0.071 0.071 0.071
you :   0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.143 0.071 0.071 0.071
from :  0.071 0.071 0.071 0.143 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071
(eos) : 0.056 0.056 0.056 0.056 0.056 0.111 0.056 0.056 0.056 0.056 0.056 0.056 0.056
Nice :  0.143 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071
meet :  0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.143 0.071 0.071 0.071 0.071 0.071
How :   0.071 0.071 0.143 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071

<--After applying Smoothing Probability Table-->
          to  you?  are  VESIT  I  Hi  am  you  from  (eos)  Nice  meet  How
to :    0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.133 0.067
you? :  0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.133 0.067 0.067 0.067
are :   0.067 0.133 0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.067
VESIT : 0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.133 0.067 0.067 0.067
I :     0.067 0.067 0.067 0.067 0.067 0.067 0.133 0.067 0.067 0.067 0.067 0.067 0.067
Hi :    0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.133 0.067 0.067 0.067
am :    0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.133 0.067 0.067 0.067 0.067
you :   0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.133 0.067 0.067 0.067
from :  0.067 0.067 0.067 0.133 0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.067
(eos) : 0.053 0.053 0.053 0.053 0.053 0.105 0.053 0.053 0.053 0.053 0.053 0.053 0.053
Nice :  0.133 0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.067
meet :  0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.133 0.067 0.067 0.067 0.067 0.067
How :   0.067 0.067 0.133 0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.067 0.067
```