How can I calculate perplexity using nltk

Asked 8 months ago Active 8 months ago Viewed 1k times

1

I try to do some process on a text. It's part of my code:

```
fp = open(train_file)
raw = fp.read()
sents = fp.readlines()
words = nltk.tokenize.word_tokenize(raw)
bigrams = ngrams(words,2, left_pad_symbol='<s>', right_pad_symbol=</s>)
fdist = nltk.FreqDist(words)
```

In the old versions of `nltk` I found this code on StackOverflow for `perplexity`

```
estimator = lambda fdist, bins: LidstoneProbDist(fdist, 0.2)
lm = NgramModel(5, train, estimator=estimator)
print("len(corpus) = %s, len(vocabulary) = %s, len(train) = %s, len(test) = %s" % (
len(corpus), len(vocabulary), len(train), len(test) ))
print("perplexity(test) =", lm.perplexity(test))
```

However, this code is no longer valid, and I didn't find any other package or function in `nltk` for this purpose. Should I implement it?

python-3.x nltk

edited Mar 1 at 9:56

asked Mar 1 at 9:48

Ahmad

4,779 4 33 67

1 Answer

4

Perplexity

Lets assume we have a model which takes as input an English sentence and gives out a probability score corresponding to how likely its is a valid English sentence. We want to determined how good this model is. A good model should give high score to valid English sentences and low score to invalid English sentences. Perplexity is a popularly used measure to quantify how "good" such a model is. If a sentence *s* contains *n* words then perplexity

$$PP(s) = p(w_1, \cdots, w_n)^{-\frac{1}{n}}$$

Modeling probability distribution p (building the model)

$p(w_1, \cdots, w_n)$ can be expanded using chain rule of probability

$$p(w_1, \cdots, w_n) = p(w_1) * p(w_2|w_1) * p(w_3|w_1, w_2) * \cdots * p(w_k|w_1, \cdots, w_{k-1})$$

So given some data (called train data) we can calculated the above conditional probabilities. However, practically it is not possible as it will requires huge amount of training data. We then make assumption to calculate

$p(w_1, \cdots, w_n)$

Assumption : All words are independent (unigram)

$$p(w_1, \cdots, w_n) = p(w_1) * p(w_2) * \cdots * p(w_n)$$

Assumption : First order Markov assumption (bigram)

Next words depends only on the previous word

$$p(w_1, w_2, w_3, \cdots w_n) = p(w_1) * p(w_2|w_1) * \cdots * p(w_n|w_{n-1})$$

Assumption : n order Markov assumption (ngram)

Next words depends only on the previous *n* words

MLE to estimate probabilities

Maximum Likelihood Estimate(MLE) is one way to estimate the individual probabilities

Unigram

$p(w) = \frac{count(w)}{count(vocab)}$ where

- count(w) is number of times the word w appears in the train data
- count(vocab) is the number of uniques words (called vocabulary) in the train data.

Bigram

$p(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_i - 1)}$ where

- count(w_{i-1}, w_i) is number of times the words w_{i-1}, w_i appear together in same sequence (bigram) in the train data

- count(w_{i-1}) is the number of times the word w_{i-1} appear in the train data. w_{i-1} is called context.

Calculating Perplexity

As we have seen above $p(s)$ is calculated by multiplying lots of small numbers and so it is not numerically stable because of limited precision of floating point numbers on a computer. Lets use the nice properties of log to simply it. We know $a^{log_a^b} = b$

$$PP(s) = 2^{log_2^{PP(s)}} = 2^{-\frac{1}{n}log(p(s))}$$

$$\text{let } l = \frac{1}{n}log(p(s))$$

$$\text{For unigram } l = \frac{1}{n}(logp(w_1) + \cdots + logp(w_n))$$

$$\text{For bigram } l = \frac{1}{n}(logp(w_1) + logp(w_2|w_1) + \cdots + logp(w_n|w_{n-1}))$$

Example: Unigram model

Train Data ["an apple", "an orange"] Vocabulary : [an, apple, orange, UNK]

MLE estimates

p(w)	MLE estimate
p(an)	2/4 = 0.5
p(apple)	1/4 = 0.25
p(orange)	1/4 = 0.25
p(ant)	0/4 = 0

For test sentence "an apple"

```
l =  (np.log2(0.5) + np.log2(0.25))/2 = -1.5
np.power(2, -l) = 2.8284271247461903
```

For test sentence "an ant"

```
l =  (np.log2(0.5) + np.log2(0))/2 = inf
```

Code

```
import nltk
from nltk.lm.preprocessing import padded_everygram_pipeline
from nltk.lm import MLE

train_sentences = ['an apple', 'an orange']
tokenized_text = [list(map(str.lower, nltk.tokenize.word_tokenize(sent)))
                  for sent in train_sentences]
n = 1
train_data, padded_vocab = padded_everygram_pipeline(n, tokenized_text)
model = MLE(n)
model.fit(train_data, padded_vocab)

test_sentences = ['an apple', 'an ant']
tokenized_text = [list(map(str.lower, nltk.tokenize.word_tokenize(sent)))
                  for sent in test_sentences]

test_data, _ = padded_everygram_pipeline(n, tokenized_text)
for test in test_data:
    print ("MLE Estimates:", [((ngram[-1], ngram[:-1]),model.score(ngram[-1],
ngram[:-1])) for ngram in test])

test_data, _ = padded_everygram_pipeline(n, tokenized_text)

for i, test in enumerate(test_data):
  print("PP({0}):{1}".format(test_sentences[i], model.perplexity(test)))
```

Example: Bigram model

Train Data: "an apple", "an orange" Padded Train Data: "(s) an apple (/s)", "(s) an orange (/s)" Vocabulary : (s), (/s) an, apple, orange, UNK

MLE estimates

p(w)	MLE estimate
p(an|s)	2/2 = 1
p(apple|an)	1/2 = 0.5
p(\s|apple)	1/1 = 1
p(ant|an)	0/1 = 0
p(\s|ant)	0

For test sentence "an apple" Padded : "(s) an apple (/s)"

```
l =  (np.log2(p(an|<s> ) + np.log2(p(apple|an) + np.log2(p(</s>|apple))/3 =
(np.log2(1) + np.log2(0.5) + np.log2(1))/3 = -0.3333
np.power(2, -l) = 1.
```

For test sentence "an ant" Padded : "(s) an ant (/s)"

```
l =  (np.log2(p(an|<s> ) + np.log2(p(ant|an) + np.log2(p(</s>|ant))/3 = inf
```

Code

```
import nltk
from nltk.lm.preprocessing import padded_everygram_pipeline
from nltk.lm import MLE
from nltk.lm import Vocabulary

train_sentences = ['an apple', 'an orange']
tokenized_text = [list(map(str.lower, nltk.tokenize.word_tokenize(sent))) for sent in
train_sentences]

n = 2
train_data = [nltk.bigrams(t,  pad_right=True, pad_left=True, left_pad_symbol="<s>",
right_pad_symbol="</s>") for t in tokenized_text]
words = [word for sent in tokenized_text for word in sent]
words.extend(["<s>", "</s>"])
padded_vocab = Vocabulary(words)
model = MLE(n)
model.fit(train_data, padded_vocab)

test_sentences = ['an apple', 'an ant']
tokenized_text = [list(map(str.lower, nltk.tokenize.word_tokenize(sent))) for sent in
test_sentences]

test_data = [nltk.bigrams(t,  pad_right=True, pad_left=True, left_pad_symbol="<s>",
right_pad_symbol="</s>") for t in tokenized_text]
for test in test_data:
    print ("MLE Estimates:", [((ngram[-1], ngram[:-1]),model.score(ngram[-1],
ngram[:-1])) for ngram in test])

test_data = [nltk.bigrams(t,  pad_right=True, pad_left=True, left_pad_symbol="<s>",
right_pad_symbol="</s>") for t in tokenized_text]
for i, test in enumerate(test_data):
  print("PP({0}):{1}".format(test_sentences[i], model.perplexity(test)))
```

answered Mar 7 at 12:35

mujjiga

5,470 2 16 24