

Language Technology

Chapter 11: Word Sequences https://link.springer.com/chapter/10.1007/978-3-031-57549-5_10

[//link.springer.com/chapter/10.1007/978-3-031-57549-5_10](https://link.springer.com/chapter/10.1007/978-3-031-57549-5_10)

Pierre Nugues

Pierre.Nugues@cs.lth.se

September 8 and 11, 2025



Word Sequences

Words have specific contexts of use.

Pairs of words like *strong* and *tea* or *powerful* and *computer* are not random associations.

Psychological linguistics tells us that it is difficult to make a difference between *writer* and *rider* without context

A listener will discard the improbable *rider of books* and prefer *writer of books*

A language model is the statistical estimate of a word sequence.

Originally developed for speech recognition

The language model component enables to predict the next word given a sequence of previous words: *the writer of books, novels, poetry, etc.* and not *the writer of hooks, nobles, poultry, ...*



Statistical Estimates

We will build a model to estimate the likelihood of sequence hypotheses:

$$P(I \text{ wanted to be a book writer})$$

and

$$P(I \text{ wanted to be a book rider})$$

and keep the highest

Same with:

$$P(\text{The buoys eat the sand which is})$$

and

$$P(\text{The boys eat the sandwiches})$$



N-Grams

The types are the distinct words of a text while the tokens are all the words or symbols.

The phrases from *Nineteen Eighty-Four*

War is peace

Freedom is slavery

Ignorance is strength

have 9 tokens and 7 types.

Unigrams are single words

Bigrams are sequences of two words

Trigrams are sequences of three words



Trigrams

| Word | Rank | More likely alternatives |
|-----------|------|---|
| We | 9 | <i>The This One Two A Three Please In</i> |
| need | 7 | <i>are will the would also do</i> |
| to | 1 | |
| resolve | 85 | <i>have know do. . .</i> |
| all | 9 | <i>the this these problems. . .</i> |
| of | 2 | <i>the</i> |
| the | 1 | |
| important | 657 | <i>document question first. . .</i> |
| issues | 14 | <i>thing point to. . .</i> |
| within | 74 | <i>to of and in that. . .</i> |
| the | 1 | |
| next | 2 | <i>company</i> |
| two | 5 | <i>page exhibit meeting day</i> |
| days | 5 | <i>weeks years pages months</i> |



Probabilistic Models of a Word Sequence

$$\begin{aligned}P(S) &= P(w_1, \dots, w_n), \\&= P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)\dots P(w_n|w_1, \dots, w_{n-1}), \\&= \prod_{i=1}^n P(w_i|w_1, \dots, w_{i-1}).\end{aligned}$$

The probability $P(\textit{It was a bright cold day in April})$ from *Nineteen Eighty-Four* corresponds to

\textit{It} to begin the sentence, then \textit{was} knowing that we have \textit{It} before, then \textit{a} knowing that we have $\textit{It was}$ before, and so on until the end of the sentence.

$$\begin{aligned}P(S) &= P(\textit{It}) \times P(\textit{was}|\textit{It}) \times P(\textit{a}|\textit{It}, \textit{was}) \times P(\textit{bright}|\textit{It}, \textit{was}, \textit{a}) \times \dots \\&\quad \times P(\textit{April}|\textit{It}, \textit{was}, \textit{a}, \textit{bright}, \dots, \textit{in}).\end{aligned}$$



Approximations

Bigrams:

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

Trigrams:

$$P(w_i | w_1, w_2, \dots, w_{i-1}) \approx P(w_i | w_{i-2}, w_{i-1}).$$

Using a trigram language model, $P(S)$ is approximated as:

$$P(S) \approx P(It) \times P(was|It) \times P(a|It, was) \times P(bright|was, a) \times \dots \\ \times P(April|day, in).$$



Counting Bigrams With Unix Tools

- ❶ `tr -cs 'A-Za-z' '\n' < input_file > token_file`
Tokenize the input and create a file with the unigrams.
- ❷ `tail +2 < token_file > next_token_file`
Create a second unigram file starting at the second word of the first tokenized file (+2).
- ❸ `paste token_file next_token_file > bigrams`
Merge the lines (the tokens) pairwise. Each line of `bigrams` contains the words at index i and $i+1$ separated with a tabulation.
- ❹ And we count the bigrams as in the previous script.



Counting Bigrams in Python

```
bigrams = [tuple(words[inx:inx + 2])  
            for inx in range(len(words) - 1)]
```

The rest of the `count_bigrams` function is nearly identical to `count_unigrams`. As input, it uses the same list of words:

```
def count_bigrams(words):  
    bigrams = [tuple(words[inx:inx + 2])  
                for inx in range(len(words) - 1)]  
    frequencies = {}  
    for bigram in bigrams:  
        if bigram in frequencies:  
            frequencies[bigram] += 1  
        else:  
            frequencies[bigram] = 1  
    return frequencies
```



Maximum Likelihood Estimate

Bigrams:

$$P_{MLE}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i)}{\sum_w C(w_{i-1}, w)} = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}.$$

Trigrams:

$$P_{MLE}(w_i|w_{i-2}, w_{i-1}) = \frac{C(w_{i-2}, w_{i-1}, w_i)}{C(w_{i-2}, w_{i-1})}.$$



Conditional Probabilities

A common mistake in computing the conditional probability $P(w_i|w_{i-1})$ is to use

$$\frac{C(w_{i-1}, w_i)}{\# \text{bigrams}}.$$

This is not correct. This formula corresponds to $P(w_{i-1}, w_i)$.
The correct estimation is

$$P_{MLE}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i)}{\sum_w C(w_{i-1}, w)} = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}.$$

Proof:

$$P(w_1, w_2) = P(w_1)P(w_2|w_1) = \frac{C(w_1)}{\# \text{words}} \times \frac{C(w_1, w_2)}{C(w_1)} = \frac{C(w_1, w_2)}{\# \text{words}}$$



Demo

<https://github.com/pnugues/pnlp/tree/main/notebooks>



Training an N -gram Model

- The model is trained on a part of the corpus: the **training set**
- It is tested on (applied to) a different part: the **test set**
- The vocabulary can be derived from the corpus, for instance the 20,000 most frequent words, or from a lexicon
- It can be closed or open
- A closed vocabulary does not accept any new word
- An open vocabulary maps the new words, either in the training or test sets, to a specific symbol, <UNK>



Probability of a Sentence: Unigrams

<s> A good deal of the literature of the past was, indeed, already being transformed in this way </s>

| w_i | $C(w_i)$ | #words | $P_{MLE}(w_i)$ |
|--------------------|----------|--------|----------------------|
| <i><s></i> | 7072 | — | |
| <i>a</i> | 2482 | 108140 | 0.023 |
| <i>good</i> | 53 | 108140 | 0.00049 |
| <i>deal</i> | 5 | 108140 | $4.62 \cdot 10^{-5}$ |
| <i>of</i> | 3310 | 108140 | 0.031 |
| <i>the</i> | 6248 | 108140 | 0.058 |
| <i>literature</i> | 7 | 108140 | $6.47 \cdot 10^{-5}$ |
| <i>of</i> | 3310 | 108140 | 0.031 |
| <i>the</i> | 6248 | 108140 | 0.058 |
| <i>past</i> | 99 | 108140 | 0.00092 |
| <i>was</i> | 2211 | 108140 | 0.020 |
| <i>indeed</i> | 17 | 108140 | 0.00016 |
| <i>already</i> | 64 | 108140 | 0.00059 |
| <i>being</i> | 80 | 108140 | 0.00074 |
| <i>transformed</i> | 1 | 108140 | $9.25 \cdot 10^{-6}$ |
| <i>in</i> | 1759 | 108140 | 0.016 |
| <i>this</i> | 264 | 108140 | 0.0024 |
| <i>way</i> | 122 | 108140 | 0.0011 |
| <i></s></i> | 7072 | 108140 | 0.065 |



Probability of a Sentence: Bigrams

<s> A good deal of the literature of the past was, indeed, already being transformed in this way </s>

| w_{i-1}, w_i | $C(w_{i-1}, w_i)$ | $C(w_{i-1})$ | $P_{MLE}(w_i w_{i-1})$ |
|--------------------------|-------------------|--------------|--------------------------|
| <i><s> a</i> | 133 | 7072 | 0.019 |
| <i>a good</i> | 14 | 2482 | 0.006 |
| <i>good deal</i> | 0 | 53 | 0.0 |
| <i>deal of</i> | 1 | 5 | 0.2 |
| <i>of the</i> | 742 | 3310 | 0.224 |
| <i>the literature</i> | 1 | 6248 | 0.0002 |
| <i>literature of</i> | 3 | 7 | 0.429 |
| <i>of the</i> | 742 | 3310 | 0.224 |
| <i>the past</i> | 70 | 6248 | 0.011 |
| <i>past was</i> | 4 | 99 | 0.040 |
| <i>was indeed</i> | 0 | 2211 | 0.0 |
| <i>indeed already</i> | 0 | 17 | 0.0 |
| <i>already being</i> | 0 | 64 | 0.0 |
| <i>being transformed</i> | 0 | 80 | 0.0 |
| <i>transformed in</i> | 0 | 1 | 0.0 |
| <i>in this</i> | 14 | 1759 | 0.008 |
| <i>this way</i> | 3 | 264 | 0.011 |
| <i>way </s></i> | 18 | 122 | 0.148 |



Sparse Data

Given a vocabulary of 20,000 types, the potential number of bigrams is $20,000^2 = 400,000,000$

With trigrams $20,000^3 = 8,000,000,000,000$

Methods:

- Laplace: add one to all counts
- Linear interpolation:

$$P_{\text{DellInterpolation}}(w_n | w_{n-2}, w_{n-1}) = \lambda_1 P_{MLE}(w_n | w_{n-2} w_{n-1}) + \lambda_2 P_{MLE}(w_n | w_{n-1}) + \lambda_3 P_{MLE}(w_n)$$

- Good-Turing: The discount factor is variable and depends on the number of times a n-gram has occurred in the corpus.
- Back-off



Laplace's Rule

$$P_{Laplace}(w_{i+1}|w_i) = \frac{C(w_i, w_{i+1}) + 1}{\sum_w (C(w_i, w) + 1)} = \frac{C(w_i, w_{i+1}) + 1}{C(w_i) + \text{Card}(V)},$$

| w_i, w_{i+1} | $C(w_i, w_{i+1}) + 1$ | $C(w_i) + \text{Card}(V)$ | $P_{Lap}(w_{i+1} w_i)$ |
|-------------------|-----------------------|---------------------------|------------------------|
| <s> a | 133 + 1 | 7072 + 8635 | 0.0085 |
| a good | 14 + 1 | 2482 + 8635 | 0.0013 |
| good deal | 0 + 1 | 53 + 8635 | 0.00012 |
| deal of | 1 + 1 | 5 + 8635 | 0.00023 |
| of the | 742 + 1 | 3310 + 8635 | 0.062 |
| the literature | 1 + 1 | 6248 + 8635 | 0.00013 |
| literature of | 3 + 1 | 7 + 8635 | 0.00046 |
| of the | 742 + 1 | 3310 + 8635 | 0.062 |
| the past | 70 + 1 | 6248 + 8635 | 0.0048 |
| past was | 4 + 1 | 99 + 8635 | 0.00057 |
| was indeed | 0 + 1 | 2211 + 8635 | 0.000092 |
| indeed already | 0 + 1 | 17 + 8635 | 0.00012 |
| already being | 0 + 1 | 64 + 8635 | 0.00011 |
| being transformed | 0 + 1 | 80 + 8635 | 0.00011 |
| transformed in | 0 + 1 | 1 + 8635 | 0.00012 |
| in this | 14 + 1 | 1759 + 8635 | 0.0014 |
| this way | 3 + 1 | 264 + 8635 | 0.00045 |
| way </s> | 18 + 1 | 122 + 8635 | 0.0022 |



Good–Turing

Laplace's rule shifts an enormous mass of probability to very unlikely bigrams. Good–Turing's estimation is more effective

Let's denote N_c the number of n-grams that occurred exactly c times in the corpus.

N_0 is the number of unseen n-grams, N_1 the number of n-grams seen once, N_2 the number of n-grams seen twice The frequency of n-grams occurring c times is re-estimated as:

$$c^* = (c + 1) \frac{E(N_{c+1})}{E(N_c)},$$

Unseen n-grams: $c^* = \frac{N_1}{N_0}$ and N-grams seen once: $c^* = \frac{2N_2}{N_1}$.



Good-Turing for *Nineteen eighty-four*

Nineteen eighty-four contains 37,365 unique bigrams and 5,820 bigram seen twice.

Its vocabulary of 8,635 words generates $8635^2 = 74,563,225$ bigrams whose 74,513,701 are unseen.

New counts:

- Unseen bigrams: $\frac{37,365}{74,513,701} = 0.0005$.
- Unique bigrams: $2 \times \frac{5820}{37,365} = 0.31$.
- Etc.

| Freq. of occ. | N_c | c^* | Freq. of occ. | N_c | c^* |
|---------------|------------|--------|---------------|-------|-------|
| 0 | 74,513,701 | 0.0005 | 5 | 719 | 3.91 |
| 1 | 37,365 | 0.31 | 6 | 468 | 4.94 |
| 2 | 5,820 | 1.09 | 7 | 330 | 6.06 |
| 3 | 2,111 | 2.02 | 8 | 250 | 6.44 |
| 4 | 1,067 | 3.37 | 9 | 179 | 8.93 |



Backoff

If there is no bigram, then use unigrams:

$$P_{\text{Backoff}}(w_i|w_{i-1}) = \begin{cases} \tilde{P}(w_i|w_{i-1}), & \text{if } C(w_{i-1}, w_i) \neq 0, \\ \alpha P(w_i), & \text{otherwise.} \end{cases}$$

Simplified backoff:

$$P_{\text{Backoff}}(w_i|w_{i-1}) = \begin{cases} P_{\text{MLE}}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}, & \text{if } C(w_{i-1}, w_i) \neq 0, \\ P_{\text{MLE}}(w_i) = \frac{C(w_i)}{\# \text{words}}, & \text{otherwise.} \end{cases}$$

The sum of probabilities is not equal to one though.



Backoff: Example

| w_{i-1}, w_i | $C(w_{i-1}, w_i)$ | $C(w_i)$ | $P_{\text{Backoff}}(w_i w_{i-1})$ |
|-------------------|-------------------|----------|-------------------------------------|
| <s> | | 7072 | — |
| <s> a | 133 | 2482 | 0.019 |
| a good | 14 | 53 | 0.006 |
| good deal | 0 | 5 | $4.62 \cdot 10^{-5}$ |
| deal of | 1 | 3310 | 0.2 |
| of the | 742 | 6248 | 0.224 |
| the literature | 1 | 7 | 0.00016 |
| literature of | 3 | 3310 | 0.429 |
| of the | 742 | 6248 | 0.224 |
| the past | 70 | 99 | 0.011 |
| past was | 4 | 2211 | 0.040 |
| was indeed | 0 | 17 | 0.00016 |
| indeed already | 0 | 64 | 0.00059 |
| already being | 0 | 80 | 0.00074 |
| being transformed | 0 | 1 | $9.25 \cdot 10^{-6}$ |
| transformed in | 0 | 1759 | 0.016 |
| in this | 14 | 264 | 0.008 |
| this way | 3 | 122 | 0.011 |
| way </s> | 18 | 7072 | 0.148 |

The figures we obtain are not probabilities. We can use the Good-Turing technique to discount the bigrams and then scale the unigram probabilities. This is the Katz backoff.



Quality of a Language Model (I)

The quality of a language model corresponds to its accuracy in predicting word sequences: $P(w_1, \dots, w_n)$: The higher, the better.

We derive the model (the statistics) from a training set and evaluate this quality on a long unseen sequence sequence: The test set.

With the n -gram approximations, we have:

$$P(w_1, \dots, w_n) = \prod_{i=1}^n P(w_i) \quad \text{Unigrams}$$

$$P(w_1, \dots, w_n) = P(w_1) \prod_{i=2}^n P(w_i | w_{i-1}) \quad \text{Bigrams}$$

$$P(w_1, \dots, w_n) = P(w_1) P(w_2 | w_1) \prod_{i=3}^n P(w_i | w_{i-2}, w_{i-1}) \quad \text{Trigrams}$$

etc.



Quality of a Language Model (II)

The probability value will depend on the length of the sequence. We take the geometric mean instead to standardize across different lengths:

$$\sqrt[n]{\prod_{i=1}^n P(w_i)} \quad \text{Unigrams}$$

$$\sqrt[n]{P(w_1) \prod_{i=2}^n P(w_i | w_{i-1})} \quad \text{Bigrams}$$

...

In practice, we use the log to compute the per word probability of a word sequence, the entropy rate:

$$H(L) = -\frac{1}{n} \log_2 P(w_1, \dots, w_n).$$

Here the lower, the better

The figures are usually presented with the perplexity metric:

$$PP(P, M) = 2^{H(L)}.$$



Mathematical Background

Entropy rate: $H_{rate} = -\frac{1}{n} \sum_{w_1, \dots, w_n \in L} P(w_1, \dots, w_n) \log_2 P(w_1, \dots, w_n),$

Cross entropy:

$$H(P, M) = -\frac{1}{n} \sum_{w_1, \dots, w_n \in L} P(w_1, \dots, w_n) \log_2 M(w_1, \dots, w_n).$$

We have:

$$\begin{aligned} H(P, M) &= \lim_{n \rightarrow \infty} -\frac{1}{n} \sum_{w_1, \dots, w_n \in L} P(w_1, \dots, w_n) \log_2 M(w_1, \dots, w_n), \\ &= \lim_{n \rightarrow \infty} -\frac{1}{n} \log_2 M(w_1, \dots, w_n). \end{aligned}$$

We compute the cross entropy on the complete word sequence of a test set, governed by P , using a bigram or trigram model, M , from a training set.



Masked Language Models

Language models we have seen are said to be **causal** or **autoregressive**:

$$\arg \max_{x_i \in V} P(x_i | x_1, x_2, \dots, x_{i-1})$$

Masked language models predict a word from a left and right context, as for instance:

A good deal of the literature of the [MASK] was indeed already being transformed in this way

from the sentence

*A good deal of the literature of the **past** was indeed already being transformed in this way*

They correspond to cloze tests in language learning (or language tests):

$$\arg \max_{x_i \in V} P(x_i | x_1, x_2, \dots, x_{i-2}, x_{i-1}, x_{i+1}, x_{i+2}, \dots, x_n)$$

Good models require a large and complex neural architecture
Transformers are an example of them.



Generating Text with a Language Model

In speech recognition, language models help predict the next word, x_i given a sequence of preceding words, x_1, x_2, \dots, x_{i-1} and an acoustic input, A

$$\arg \max_{x_i \in V} P(x_i | x_1, x_2, \dots, x_{i-1}, A).$$

This results eventually in a more likely sequence

We can remove the speech input, :

$$P(x_i | x_1, x_2, \dots, x_{i-1})$$

simply add the predicted word to the existing sequence,

$$P(x_{i+1} | x_1, x_2, \dots, x_{i-1}, x_i)$$

and repeat the operation. We will then generate text



Generating Text with Bigrams

Let us use bigrams to simplify:

$$P(x_i|x_{i-1})$$

And let us first keep the same distribution in the training set and the generated sequence

Starting from the last word in the sequence, say *Hector*, we estimate:

$$P(x|hector)$$

```
[(('hector', 'and'), 0.11666666666666667),  
 (('hector', 'son'), 0.052083333333333336),  
 (('hector', 's'), 0.04791666666666667),  
 (('hector', 'was'), 0.03125),  
 (('hector', 'in'), 0.022916666666666665),  
 ...]
```



Drawing the Next Word

We will use `np.random.multinomial()` to draw the next word with the same distribution.

```
np.random.multinomial(1, [0.3, 0.5, 0.2])
```

returns a unit vector following the distribution of the second argument.
Repeating:

```
np.random.multinomial(1, [0.3, 0.5, 0.2])
```

produces:

```
[0 0 1]
```

```
[0 1 0]
```

```
[0 1 0]
```

```
...
```

In the end, $[1, 0, 0]$ represents 30% of the samples, $[0, 1, 0]$, 50% and $[0, 0, 1]$, 20%.



Generating Text

We define a start word, for instance *Hector*

We select the next word according to the multinomial distribution; we use this second word to select the third and so on; This generates a text like this one:

hector has slain noble agenor away from your prophets the bravest of the trojans while achilles was choking him roughly away and as all alone for evermore hades that is not fallen...



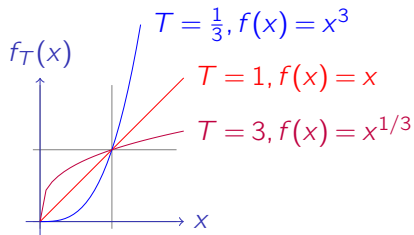
Transforming the Distribution

We can transform the distribution over the second word in the bigram to make the generation more deterministic or more random.

We will follow Chollet in *Deep Learning with Python*, 2nd ed., pp. 369 and 373. with a temperature: $f_T(x) = e^{\frac{\log x}{T}}$.

This is equivalent to a power function:

$$f_T(x) = x^{\frac{1}{T}}$$



Code

```
def power_transform(distribution, T=0.5):  
    new_dist = np.power(distribution, 1/T)  
    return new_dist / np.sum(new_dist)
```

Demo: <https://github.com/pnugues/pnlp/tree/main/notebooks>



Results

With a temperature of 3.0, we obtain:

*hector fearless of perimedes leader of saturn devise evil for junos
if ever whereas in that forms on him spear or be from over land
prian would prove me honour*

With 0.5:

*hector and he was lying dream went up to the achaeans will be
bought nor of the shield of his father jove in the trojans and the
son of the achaeans and the argives and the trojans and the body
of the achaeans if you are you have been dearest to*



Other Statistical Formulas

- Mutual information (The strength of an association):

$$I(w_i, w_j) = \log_2 \frac{P(w_i, w_j)}{P(w_i)P(w_j)} \approx \log_2 \frac{N \cdot C(w_i, w_j)}{C(w_i)C(w_j)}.$$

- T-score (The confidence of an association):

$$\begin{aligned} t(w_i, w_j) &= \frac{\text{mean}(P(w_i, w_j)) - \text{mean}(P(w_i))\text{mean}(P(w_j))}{\sqrt{\sigma^2(P(w_i, w_j)) + \sigma^2(P(w_i)P(w_j))}}, \\ &\approx \frac{C(w_i, w_j) - \frac{1}{N}C(w_i)C(w_j)}{\sqrt{C(w_i, w_j)}}. \end{aligned}$$



T-Scores with Word set

| Word | Frequency | Bigram set + word | <i>t</i> -score |
|------------|-----------|-------------------|-----------------|
| <i>up</i> | 134,882 | 5512 | 67.980 |
| <i>a</i> | 1,228,514 | 7296 | 35.839 |
| <i>to</i> | 1,375,856 | 7688 | 33.592 |
| <i>off</i> | 52,036 | 888 | 23.780 |
| <i>out</i> | 12,3831 | 1252 | 23.320 |

Source: Bank of English



Mutual Information with Word *surgery*

| Word | Frequency | Bigram word + surgery | Mutual info |
|-----------------------|-----------|-----------------------|-------------|
| <i>arthroscopic</i> | 3 | 3 | 11.822 |
| <i>pioneering</i> | 3 | 3 | 11.822 |
| <i>reconstructive</i> | 14 | 11 | 11.474 |
| <i>refractive</i> | 6 | 4 | 11.237 |
| <i>rhinoplasty</i> | 5 | 3 | 11.085 |

Source: Bank of English



Mutual Information in Python

```
def mutual_info(words, freq_unigrams, freq_bigrams):  
    mi = {}  
    factor = len(words) * len(words) / (len(words) - 1)  
    for bigram in freq_bigrams:  
        mi[bigram] = (  
            math.log(factor * freq_bigrams[bigram] /  
                    (freq_unigrams[bigram[0]] *  
                     freq_unigrams[bigram[1]]), 2))  
    return mi
```



T-Scores in Python

```
def t_scores(words, freq_unigrams, freq_bigrams):  
    ts = {}  
    for bigram in freq_bigrams:  
        ts[bigram] = ((freq_bigrams[bigram] -  
                        freq_unigrams[bigram[0]] *  
                        freq_unigrams[bigram[1]] /  
                        len(words)) /  
                      math.sqrt(freq_bigrams[bigram]))  
  
    return ts
```

