Language Technology

Chapter 10: Word Sequences 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:

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

and

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

and keep the highest
Same with:

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

and

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

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

Probabilistic Models of a Word Sequence

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

The probability $P(\textit{It was a bright cold day in April})$ from *Nineteen Eighty-Four* corresponds to
It to begin the sentence, then *was* knowing that we have *It* before, then *a* knowing that we have *It was* before, and so on until the end of the sentence.

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

Approximations

Bigrams:

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

Trigrams:

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

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

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

Counting Bigrams With Unix Tools

1. ```
   tr -cs 'A-Za-z' '\n' < input_file > token_file
   ```
   Tokenize the input and create a file with the unigrams.
2. ```
   tail +2 < token_file > next_token_file
   ```
   Create a second unigram file starting at the second word of the first tokenized file (+2).
3. ```
   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.
4. 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

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

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:

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

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	$P_{MLE}(w_i)$
<s>	7072	–	
a	2482	108140	0.023
good	53	108140	0.00049
deal	5	108140	$4.62\ 10^{-5}$
of	3310	108140	0.031
the	6248	108140	0.058
literature	7	108140	$6.47\ 10^{-5}$
of	3310	108140	0.031
the	6248	108140	0.058
past	99	108140	0.00092
was	2211	108140	0.020
indeed	17	108140	0.00016
already	64	108140	0.00059
being	80	108140	0.00074
transformed	1	108140	$9.25\ 10^{-6}$
in	1759	108140	0.016
this	264	108140	0.0024
way	122	108140	0.0011
</s>	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	$C(w_{i-1}, w_i)$	$C(w_{i-1})$	$P_{MLE}(w_i \mid w_{i-1})$
<s> *a*	133	7072	0.019
a good	14	2482	0.006
good deal	0	53	0.0
deal of	1	5	0.2
of the	742	3310	0.224
the literature	1	6248	0.0002
literature of	3	7	0.429
of the	742	3310	0.224
the past	70	6248	0.011
past was	4	99	0.040
was indeed	0	2211	0.0
indeed already	0	17	0.0
already being	0	64	0.0
being transformed	0	80	0.0
transformed in	0	1	0.0
in this	14	1759	0.008
this way	3	264	0.011
way </s>	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:

$$\begin{aligned} P_{DelInterpolation}(w_n|w_{n-2}, w_{n-1}) &= \lambda_1 P_{MLE}(w_n|w_{n-2}w_{n-1}) + \\ & \lambda_2 P_{MLE}(w_n|w_{n-1}) + \lambda_3 P_{MLE}(w_n) \end{aligned}$$

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

Laplace's Rule

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

w_i, w_{i+1}	$C(w_i, w_{i+1}) + 1$	$C(w_i) + Card(V)$	$P_{Lap}(w_{i+1}\|w_i)$
<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, etc.

The frequency of n-grams occurring c times is re-estimated as:

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

More generally:

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

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

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

The sum of probabilities is not equal to one though.

Backoff: Example

w_{i-1}, w_i	$C(w_{i-1}, w_i)$		$C(w_i)$	$P_{Backoff}(w_i \vert w_{i-1})$
<s>			7072	—
<s> *a*	133		2482	0.019
a good	14		53	0.006
good deal	0	backoff	5	$4.62\ 10^{-5}$
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	backoff	17	0.00016
indeed already	0	backoff	64	0.00059
already being	0	backoff	80	0.00074
being transformed	0	backoff	1	$9.25\ 10^{-6}$
transformed in	0	backoff	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: $P(w_1,...,w_n)$: 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:

$$
\begin{array}{rcll}
P(w_1,...,w_n) & = & \prod_{i=1}^{n} P(w_i) & \text{Unigrams} \\
P(w_1,...,w_n) & = & P(w_1)\prod_{i=2}^{n} P(w_i|w_{i-1}) & \text{Bigrams} \\
P(w_1,...,w_n) & = & P(w_1)P(w_2|w_1)\prod_{i=3}^{n} P(w_i|w_{i-2},w_{i-1}) & \text{Trigrams}
\end{array}
$$

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:

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

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

...

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

Unigrams: $\frac{1}{n}\sum_{i=1}^{n} \log_2 P(w_i)$

Bigrams: $\frac{1}{n}(\log_2 P(w_1) + \sum_{i=2}^{n} \log_2 P(w_i|w_{i-1}))$

...

Perplexity

Mean sequence probabilities are usually small and difficult to understand
The figures are usually presented with the **perplexity** metric
Perplexity is simply the inverse of the geometric mean of the sequence probabiliy:

$$\begin{aligned} PP &= \frac{1}{\sqrt[n]{P(w_1, w_2, \ldots, w_n)}}, \\ &= \frac{1}{\sqrt[n]{\prod_{i=1}^{n} P(w_i | w_{j<i})}}. \end{aligned}$$

Here the lower, the better
Again, in practice, we use the log of this value called the entropy rate:

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

We find the perplexity with the formula:

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

Mathematical Background

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

Cross entropy:

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

We have:

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

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**:

$$\arg\max_{x_i inV} P(x_i | x_1, x_2, \ldots, x_{i-1})$$

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):

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

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, $x_1, x_2, \ldots, x_{i-1}$ and an acoustic input, A

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

This results eventually in a more likely sequence
We can remove the speech input, :

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

simply add the predicted word to the existing sequence,

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

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'), 0.05208333333333336),
(('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*

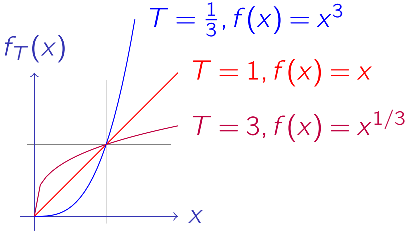
We select the next word according to the multinomial distribution; we use this second word to select the third an so on; This generates a text like this one:

> *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 juno if ever whereas in that forms on him spear or be from over land priam 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):

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

T-Scores with Word *set*

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

Source: Bank of English

Mutual Information with Word *surgery*

Word	Frequency	Bigram word + surgery	Mutual info
arthroscopic	3	3	11.822
pioneeing	3	3	11.822
reconstructive	14	11	11.474
refractive	6	4	11.237
rhinoplasty	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
```