# Language Technology
# http://cs.lth.se/edan20/
## Chapter 5: Counting Words

Pierre Nugues

Pierre.Nugues@cs.lth.se
http://cs.lth.se/pierre_nugues/

September 7, 2020

# Text Segmentation

Figure: Latin inscriptions on the *lapis niger*. *Corpus inscriptionum latinarum*, CIL I, 1. Picture from Wikipedia

# Getting the Words from a Text: Tokenization

Arrange a list of characters:

`[l, i, s, t, ' ', o, f, ' ', c, h, a, r, a, c, t, e, r, s]`

into words:

`[list, of, characters]`

Sometimes tricky:

- Dates: 28/02/96
- Numbers: 9,812.345 (English), 9 812,345 (French and German) 9.812,345 (Old fashioned French)
- Abbreviations: km/h, m.p.h.,
- Acronyms: S.N.C.F.

Tokenizers use rules (or regexes) or statistical methods.

# Tokenizing in Python: Using the Boundaries

## Simple program

```
import re

one_token_per_line = re.sub('\s+', '\n', text)
```

## Punctuation

```
import regex as re

spaced_tokens = re.sub('([\p{S}\p{P}])', r' \1 ', text)
one_token_per_line = re.sub('\s+', '\n', spaced_tokens)
```

# Tokenizing in Python: Using the Content

## Simple program

```
import regex as re

re.findall('\p{L}+', text)
```

## Punctuation

```
spaced_tokens = re.sub('([\p{S}\p{P}])', r' \1 ', text)
re.findall('[\p{S}\p{P}\p{L}]+', spaced_tokens)
```

# Improving Tokenization

The tokenization algorithm is word-based and defines a content

It does not work on nomenclatures such as Item #N23-SW32A, dates, or numbers

Instead it is possible to improve it using a boundary-based strategy with spaces (using for instance `\s`) and punctuation

But punctuation signs like commas, dots, or dashes can also be parts of tokens

Possible improvements using microgrammars

At some point, need of a dictionary:

*Can't* → can n't, *we'll* → we 'll

*J'aime* → j' aime but *aujourd'hui*

# Sentence Segmentation

As for tokenization, segmenters use either rules (or regexes) or statistical methods.

Grefenstette and Tapanainen (1994) used the Brown corpus and experimented increasingly complex rules

Most simple rule: a period corresponds to a sentence boundary: 93.20% correctly segmented

Recognizing numbers:

```
[0-9]+(\/[0-9]+)+                    Fractions, dates
([+\-])?[0-9]+(\.)?[0-9]*%           Percent
([0-9]+,?)+(\.[0-9]+|[0-9]+)*        Decimal numbers
```

93.78% correctly segmented

# Abbreviations

Common patterns (Grefenstette and Tapanainen 1994):

- single capitals: *A., B., C.,*
- letters and periods: *U.S. i.e. m.p.h.,*
- capital letter followed by a sequence of consonants: *Mr. St. Assn.*

| **Regex** | **Correct** | **Errors** | **Full stop** |
|---|---|---|---|
| `[A-Za-z]\.` | 1,327 | 52 | 14 |
| `[A-Za-z]\.([A-Za-z0-9]\.)+` | 570 | 0 | 66 |
| `[A-Z][bcdfghj-np-tvxz]+\.` | 1,938 | 44 | 26 |
| **Totals** | 3,835 | 96 | 106 |

Correct segmentation increases to 97.66%
With an abbreviation dictionary to 99.07%

# Counting Words With Unix Tools

1. `tr -cs 'A-Za-z' '\n' <input_file |`
   Tokenize the text in `input_file`, where `tr` behaves like Perl `tr`: We have one word per line and the output is passed to the next command.
2. `tr 'A-Z' 'a-z' |`
   Translate the uppercase characters into lowercase letters and pass the output to the next command.
3. `sort |`
   Sort the words. The identical words will be grouped in adjacent lines.
4. `uniq -c |`
   Remove repeated lines. The identical adjacent lines will be replaced with one single line. Each unique line in the output will be preceded by the count of its duplicates in the input file (`-c`).
5. `sort -rn |`
   Sort in the reverse (`-r`) numeric (`-n`) order: Most frequent words first.
6. `head -5`
   Print the five first lines of the file (the five most frequent words).

# Counting Words in Python

```
def tokenize(text):
    words = re.findall('\p{L}+', text)
    return words

def count_unigrams(words):
    frequency = {}
    for word in words:
        if word in frequency:
            frequency[word] += 1
        else:
            frequency[word] = 1
    return frequency
```

# Counting Words in Python (Cont'd)

```
if __name__ == '__main__':
    text = sys.stdin.read().lower()
    words = tokenize(text)
    frequency = count_unigrams(words)
    for word in sorted(frequency.keys()):
        print(word, '\t', frequency[word])
```

# Posting Lists

Many websites, such as Wikipedia, index their texts using an inverted index.
Each word in the dictionary is linked to a posting list that gives all the documents where this word occurs and its positions in a document.

**Collection**

D1: Chrysler plans new investments in Latin America.

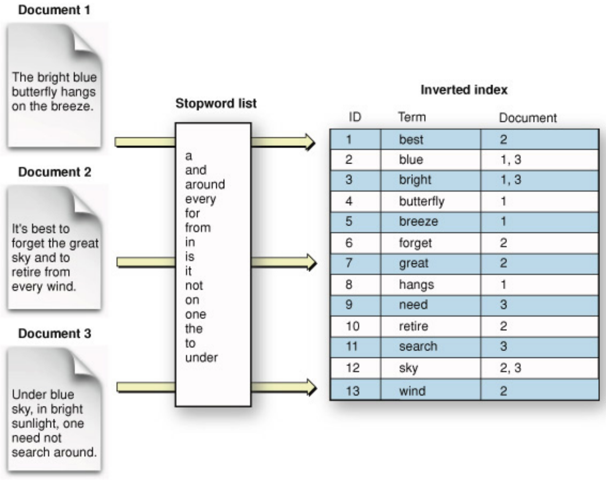
D2: Chrysler plans major investments in Mexico.

**Index**

| **Words** | **Posting lists** |
|---|---|
| *America* | (D1, 7) |
| *Chrysler* | (D1, 1) → (D2, 1) |
| *in* | (D1, 5) → (D2, 5) |
| *investments* | (D1, 4) → (D2, 4) |
| *Latin* | (D1, 6) |
| *major* | (D2, 3) |
| *Mexico* | (D2, 6) |
| *new* | (D1, 3) |
| *plans* | (D1, 2) → (D2, 2) |

Lucene is a high quality open-source indexer.

# Inverted Index (Source Apple)




| ID | Term | Document |
|---|---|---|
| 1 | best | 2 |
| 2 | blue | 1, 3 |
| 3 | bright | 1, 3 |
| 4 | butterfly | 1 |
| 5 | breeze | 1 |
| 6 | forget | 2 |
| 7 | great | 2 |
| 8 | hangs | 1 |
| 9 | need | 3 |
| 10 | retire | 2 |
| 11 | search | 3 |
| 12 | sky | 2, 3 |
| 13 | wind | 2 |

http://developer.apple.com/library/mac/documentation/UserExperience/Conceptual/SearchKitConcepts/index.html

# Information Retrieval: The Vector Space Model

The vector space model represents a document in a space of words.

| **Documents** **Words** | $w_1$ | $w_2$ | $w_3$ | ... | $w_m$ |
|---|---|---|---|---|---|
| $D_1$ | $C(w_1, D_1)$ | $C(w_2, D_1)$ | $C(w_3, D_1)$ | ... | $C(w_m, D_1)$ |
| $D_2$ | $C(w_1, D_2)$ | $C(w_2, D_2)$ | $C(w_3, D_2)$ | ... | $C(w_m, D_2)$ |
| ... | | | | | |
| $D_n$ | $C(w_1, D_n)$ | $C(w_2, D_n)$ | $C(w_3, D_n)$ | ... | $C(w_m, D_n)$ |

It was created for information retrieval to compute the similarity of two documents or to match a document and a query.
We compute the similarity of two documents through their dot product.

# The Vector Space Model: Example

A collection of two documents D1 and D2:

D1: Chrysler plans new investments in Latin America.

D2: Chrysler plans major investments in Mexico.

The vectors representing the two documents:

| D. | america | chrysler | in | investments | latin | major | mexico | new | plans |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 2 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |

The vector space model represents documents as bags of words (BOW) that do not take the word order into account.

The dot product is $\mathbf{D1} \cdot \mathbf{D2} = 0 + 1 + 1 + 1 + 0 + 0 + 0 + 0 + 1 = 4$

Their cosine is $\frac{\mathbf{D1} \cdot \mathbf{D2}}{||\mathbf{D1}||.||\mathbf{D2}||} = \frac{4}{\sqrt{7}.\sqrt{6}} = 0.62$

# Giving a Weight

Word clouds give visual weights to words


Image: Courtesy of Jonas Wisbrant

# $TF \times IDF$

The frequency alone might be misleading

Document coordinates are in fact $tf \times idf$: Term frequency by inverted document frequency.

Term frequency $tf_{i,j}$: frequency of term $j$ in document $i$

Inverted document frequency: $idf_j = \log(\frac{N}{n_j})$

# Document Similarity

Documents are vectors where coordinates could be the count of each word: $\mathbf{d} = (C(w_1), C(w_2), C(w_3), ..., C(w_n))$

The similarity between two documents or a query and a document is given by their cosine:

$$\cos(\mathbf{q}, \mathbf{d}) = \frac{\sum_{i=1}^{n} q_i d_i}{\sqrt{\sum_{i=1}^{n} q_i^2}\sqrt{\sum_{i=1}^{n} d_i^2}}.$$

# Dimension Reduction

One-hot encoding of TFIDF encoding can produce very long vectors: Imagine a vocabulary one one million words per language with 100 languages.

A solution is to produce dense vectors also called word embeddings using a dimension reduction

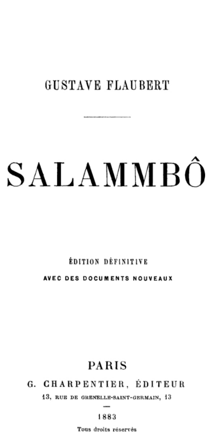
This reduction is very close to principal component analysis or singular value decomposition

It can be automatically obtained through training or initialized with pretrained vectors

# A Data Set: *Salammbô*

A corpus is a collection – a body – of texts.

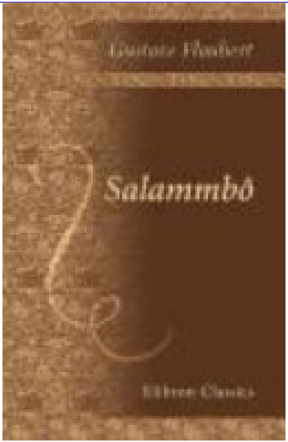
| French original | English translation |
|---|---|



GUSTAVE FLAUBERT

SALAMMBÔ

ÉDITION DÉFINITIVE
AVEC DES DOCUMENTS NOUVEAUX

PARIS
G. CHARPENTIER, ÉDITEUR
13, RUE DE GRENELLE-SAINT-GERMAIN, 13
—
1883
Tous droits réservés




# Supervised Learning

## Letter counts from *Salammbô*

| Chapter | French # characters | French # A | English # characters | English # A |
|---|---|---|---|---|
| Chapter 1 | 36,961 | 2,503 | 35,680 | 2,217 |
| Chapter 2 | 43,621 | 2,992 | 42,514 | 2,761 |
| Chapter 3 | 15,694 | 1,042 | 15,162 | 990 |
| Chapter 4 | 36,231 | 2,487 | 35,298 | 2,274 |
| Chapter 5 | 29,945 | 2,014 | 29,800 | 1,865 |
| Chapter 6 | 40,588 | 2,805 | 40,255 | 2,606 |
| Chapter 7 | 75,255 | 5,062 | 74,532 | 4,805 |
| Chapter 8 | 37,709 | 2,643 | 37,464 | 2,396 |
| Chapter 9 | 30,899 | 2,126 | 31,030 | 1,993 |
| Chapter 10 | 25,486 | 1,784 | 24,843 | 1,627 |
| Chapter 11 | 37,497 | 2,641 | 36,172 | 2,375 |
| Chapter 12 | 40,398 | 2,766 | 39,552 | 2,560 |
| Chapter 13 | 74,105 | 5,047 | 72,545 | 4,597 |
| Chapter 14 | 76,725 | 5,312 | 75,352 | 4,871 |
| Chapter 15 | 18,317 | 1,215 | 18,031 | 1,119 |

Data set: `https://github.com/pnugues/ilppp/tree/master/programs/ch04/salammbo`

# Principal Component Analysis

We will use a small dataset to explain principal component analysis: The characters in *Salammbô*

| Ch. | 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x' | 'y' | 'z' | 'à' | 'â' |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 01_fr | 2503 | 365 | 857 | 1151 | 4312 | 264 | 349 | 295 | 1945 | 65 | 4 | 1946 | 726 | 1896 | 1372 | 789 | 248 | 1948 | 2996 | 1938 | 1792 | 414 | 0 | 129 | 94 | 20 | 128 | 36 |
| 02_fr | 2992 | 391 | 1006 | 1388 | 4993 | 319 | 360 | 350 | 2345 | 81 | 6 | 2128 | 823 | 2308 | 1560 | 977 | 281 | 2376 | 3454 | 2411 | 2069 | 499 | 0 | 175 | 89 | 23 | 136 | 50 |
| 03_fr | 1042 | 152 | 326 | 489 | 1785 | 136 | 122 | 126 | 784 | 41 | 7 | 816 | 397 | 778 | 612 | 315 | 102 | 792 | 1174 | 856 | 707 | 147 | 0 | 42 | 31 | 7 | 39 | 9 |
| 04_fr | 2487 | 303 | 864 | 1137 | 4158 | 314 | 331 | 287 | 2028 | 57 | 3 | 1796 | 722 | 1958 | 1318 | 773 | 274 | 2000 | 2792 | 2031 | 1734 | 422 | 0 | 138 | 81 | 27 | 110 | 43 |
| 05_fr | 2014 | 268 | 645 | 949 | 3394 | 223 | 215 | 242 | 1617 | 67 | 3 | 1513 | 651 | 1547 | 1053 | 672 | 166 | 1601 | 2192 | 1736 | 1396 | 315 | 1 | 83 | 67 | 18 | 90 | 67 |
| 06_fr | 2805 | 368 | 910 | 1266 | 4535 | 332 | 384 | 378 | 2219 | 97 | 3 | 1900 | 841 | 2179 | 1569 | 868 | 285 | 2205 | 3065 | 2293 | 1895 | 453 | 0 | 151 | 80 | 39 | 131 | 42 |
| 07_fr | 5062 | 706 | 1770 | 2398 | 8512 | 623 | 622 | 620 | 4018 | 126 | 19 | 3726 | 1596 | 3851 | 2823 | 1532 | 468 | 4015 | 5634 | 4116 | 3518 | 844 | 0 | 272 | 148 | 71 | 246 | 50 |
| 08_fr | 2643 | 325 | 869 | 1085 | 4229 | 307 | 317 | 359 | 2102 | 85 | 4 | 1857 | 811 | 2041 | 1367 | 833 | 239 | 2132 | 2814 | 2134 | 1788 | 437 | 0 | 135 | 64 | 30 | 130 | 43 |
| 09_fr | 2126 | 289 | 771 | 920 | 3599 | 278 | 289 | 279 | 1805 | 52 | 6 | 1499 | 619 | 1711 | 1130 | 651 | 187 | 1719 | 2404 | 1763 | 1448 | 348 | 0 | 119 | 58 | 20 | 90 | 24 |
| 10_fr | 1784 | 249 | 546 | 805 | 3002 | 179 | 202 | 215 | 1319 | 60 | 5 | 1462 | 598 | 1246 | 922 | 557 | 172 | 1242 | 1769 | 1423 | 1191 | 270 | 0 | 65 | 61 | 11 | 73 | 18 |
| 11_fr | 2641 | 381 | 817 | 1078 | 4306 | 263 | 277 | 330 | 1985 | 114 | 0 | 1886 | 900 | 1966 | 1356 | 763 | 230 | 1912 | 2564 | 2218 | 1737 | 425 | 0 | 114 | 61 | 25 | 101 | 40 |
| 12_fr | 2766 | 373 | 935 | 1237 | 4618 | 329 | 350 | 349 | 2273 | 65 | 2 | 1955 | 812 | 2285 | 1419 | 865 | 272 | 2276 | 3131 | 2274 | 1923 | 455 | 0 | 149 | 98 | 37 | 129 | 33 |
| 13_fr | 5047 | 725 | 1730 | 2273 | 8678 | 648 | 566 | 642 | 3940 | 140 | 22 | 3746 | 1597 | 3984 | 2736 | 1550 | 425 | 4081 | 5599 | 4387 | 3480 | 767 | 0 | 288 | 119 | 41 | 209 | 55 |
| 14_fr | 5312 | 689 | 1754 | 2149 | 8870 | 628 | 630 | 673 | 4278 | 143 | 2 | 3780 | 1610 | 4255 | 2713 | 1599 | 512 | 4271 | 5770 | 4467 | 3697 | 914 | 0 | 283 | 145 | 41 | 224 | 75 |
| 15_fr | 1215 | 173 | 402 | 582 | 2195 | 150 | 134 | 148 | 969 | 27 | 6 | 950 | 387 | 906 | 697 | 417 | 103 | 985 | 1395 | 1037 | 893 | 206 | 0 | 63 | 36 | 3 | 48 | 20 |
| 01_en | 2217 | 451 | 729 | 1316 | 3967 | 596 | 662 | 2060 | 1823 | 22 | 200 | 1204 | 656 | 1851 | 1897 | 525 | 19 | 1764 | 1942 | 2547 | 704 | 258 | 653 | 29 | 401 | 18 | 0 | 0 |
| 02_en | 2761 | 551 | 777 | 1548 | 4543 | 685 | 769 | 2530 | 2163 | 13 | 284 | 1319 | 829 | 2218 | 2237 | 606 | 21 | 2019 | 2411 | 3083 | 861 | 295 | 769 | 37 | 475 | 31 | 0 | 0 |
| 03_en | 990 | 183 | 271 | 557 | 1570 | 279 | 253 | 875 | 783 | 4 | 82 | 520 | 333 | 816 | 828 | 194 | 13 | 711 | 864 | 1048 | 298 | 94 | 254 | 8 | 145 | 15 | 0 | 0 |
| 04_en | 2274 | 454 | 736 | 1315 | 3814 | 595 | 559 | 1978 | 1835 | 22 | 198 | 1073 | 690 | 1771 | 1865 | 514 | 33 | 1726 | 1918 | 2704 | 745 | 245 | 663 | 60 | 467 | 19 | 0 | 0 |
| 05_en | 1865 | 400 | 553 | 1135 | 3210 | 515 | 525 | 1693 | 1482 | 7 | 153 | 949 | 571 | 1468 | 1586 | 517 | 17 | 1357 | 1646 | 2178 | 663 | 194 | 568 | 26 | 330 | 33 | 0 | 0 |
| 06_en | 2606 | 518 | 797 | 1509 | 4237 | 687 | 669 | 2254 | 2097 | 26 | 216 | 1239 | 763 | 2174 | 2231 | 613 | 25 | 1931 | 2192 | 2955 | 899 | 277 | 733 | 49 | 464 | 37 | 0 | 0 |
| 07_en | 4805 | 913 | 1521 | 2681 | 7834 | 1366 | 1163 | 4379 | 3838 | 42 | 416 | 2434 | 1461 | 3816 | 4091 | 1040 | 39 | 3674 | 4060 | 5369 | 1552 | 465 | 1332 | 74 | 843 | 52 | 0 | 0 |
| 08_en | 2396 | 431 | 702 | 1416 | 4014 | 621 | 624 | 2171 | 2011 | 24 | 216 | 1152 | 748 | 2085 | 1947 | 527 | 33 | 1915 | 1966 | 2765 | 789 | 266 | 695 | 65 | 379 | 24 | 0 | 0 |
| 09_en | 1993 | 408 | 653 | 1096 | 3373 | 575 | 517 | 1766 | 1648 | 16 | 146 | 861 | 629 | 1728 | 1698 | 442 | 20 | 1561 | 1626 | 2442 | 683 | 208 | 560 | 25 | 328 | 18 | 0 | 0 |
| 10_en | 1627 | 359 | 451 | 933 | 2690 | 477 | 409 | 1475 | 1196 | 7 | 131 | 789 | 506 | 1266 | 1369 | 325 | 23 | 1211 | 1344 | 1759 | 502 | 181 | 410 | 31 | 255 | 20 | 0 | 0 |
| 11_en | 2375 | 437 | 643 | 1364 | 3790 | 610 | 644 | 2217 | 1830 | 16 | 217 | 1122 | 799 | 1833 | 1948 | 486 | 23 | 1720 | 1945 | 2424 | 767 | 246 | 632 | 20 | 457 | 39 | 0 | 0 |
| 12_en | 2560 | 489 | 757 | 1566 | 4331 | 677 | 650 | 2348 | 2033 | 28 | 234 | 1102 | 746 | 2125 | 2105 | 581 | 32 | 1939 | 2152 | 3046 | 750 | 278 | 721 | 35 | 418 | 40 | 0 | 0 |
| 13_en | 4597 | 987 | 1462 | 2689 | 7963 | 1254 | 1201 | 4278 | 3634 | 39 | 432 | 2281 | 1493 | 3774 | 3911 | 1099 | 49 | 3577 | 3894 | 5540 | 1379 | 437 | 1374 | 77 | 673 | 49 | 0 | 0 |
| 14_en | 4871 | 948 | 1439 | 2799 | 8179 | 1335 | 1140 | 4534 | 3829 | 36 | 427 | 2218 | 1534 | 4053 | 3989 | 1019 | 36 | 3689 | 3946 | 5858 | 1490 | 539 | 1377 | 90 | 856 | 49 | 0 | 0 |
| 15_en | 1119 | 229 | 335 | 683 | 1994 | 323 | 281 | 1108 | 912 | 9 | 112 | 579 | 351 | 924 | 1004 | 305 | 9 | 863 | 997 | 1330 | 310 | 108 | 330 | 14 | 150 | 9 | 0 | 0 |

Table: Character counts per chapter, where the fr and en suffixes designate the language, either French or English

Each chapter is modeled by a vector of characters.

# Character Counts

| | French | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 |
| a | 2503 | 2992 | 1042 | 2487 | 2014 | 2805 | 5062 | 2643 | 2126 | 1784 | 2641 | 2766 | 5047 | 5312 | 1215 |
| b | 365 | 391 | 152 | 303 | 268 | 368 | 706 | 325 | 289 | 249 | 381 | 373 | 725 | 689 | 173 |
| c | 857 | 1006 | 326 | 864 | 645 | 910 | 1770 | 869 | 771 | 546 | 817 | 935 | 1730 | 1754 | 402 |
| d | 1151 | 1388 | 489 | 1137 | 949 | 1266 | 2398 | 1085 | 920 | 805 | 1078 | 1237 | 2273 | 2149 | 582 |
| e | 4312 | 4993 | 1785 | 4158 | 3394 | 4535 | 8512 | 4229 | 3599 | 3002 | 4306 | 4618 | 8678 | 8870 | 2195 |
| f | 264 | 319 | 136 | 314 | 223 | 332 | 623 | 307 | 278 | 179 | 263 | 329 | 648 | 628 | 150 |
| g | 349 | 360 | 122 | 331 | 215 | 384 | 622 | 317 | 289 | 202 | 277 | 350 | 566 | 630 | 134 |
| h | 295 | 350 | 126 | 287 | 242 | 378 | 620 | 359 | 279 | 215 | 330 | 349 | 642 | 673 | 148 |
| i | 1945 | 2345 | 784 | 2028 | 1617 | 2219 | 4018 | 2102 | 1805 | 1319 | 1985 | 2273 | 3940 | 4278 | 969 |
| j | 65 | 81 | 41 | 57 | 67 | 97 | 126 | 85 | 52 | 60 | 114 | 65 | 140 | 143 | 27 |
| k | 4 | 6 | 7 | 3 | 3 | 3 | 19 | 4 | 6 | 5 | 0 | 2 | 22 | 2 | 6 |
| l | 1946 | 2128 | 816 | 1796 | 1513 | 1900 | 3726 | 1857 | 1499 | 1462 | 1886 | 1955 | 3746 | 3780 | 950 |
| m | 726 | 823 | 397 | 722 | 651 | 841 | 1596 | 811 | 619 | 598 | 900 | 812 | 1597 | 1610 | 387 |
| n | 1896 | 2308 | 778 | 1958 | 1547 | 2179 | 3851 | 2041 | 1711 | 1246 | 1966 | 2285 | 3984 | 4255 | 906 |
| o | 1372 | 1560 | 612 | 1318 | 1053 | 1569 | 2823 | 1367 | 1130 | 922 | 1356 | 1419 | 2736 | 2713 | 697 |
| p | 789 | 977 | 315 | 773 | 672 | 868 | 1532 | 833 | 651 | 557 | 763 | 865 | 1550 | 1599 | 417 |
| q | 248 | 281 | 102 | 274 | 166 | 285 | 468 | 239 | 187 | 172 | 230 | 272 | 425 | 512 | 103 |
| r | 1948 | 2376 | 792 | 2000 | 1601 | 2205 | 4015 | 2132 | 1719 | 1242 | 1912 | 2276 | 4081 | 4271 | 985 |
| s | 2996 | 3454 | 1174 | 2792 | 2192 | 3065 | 5634 | 2814 | 2404 | 1769 | 2564 | 3131 | 5599 | 5770 | 1395 |
| t | 1938 | 2411 | 856 | 2031 | 1736 | 2293 | 4116 | 2134 | 1763 | 1423 | 2218 | 2274 | 4387 | 4467 | 1037 |
| u | 1792 | 2069 | 707 | 1734 | 1396 | 1895 | 3518 | 1788 | 1448 | 1191 | 1737 | 1923 | 3480 | 3697 | 893 |
| v | 414 | 499 | 147 | 422 | 315 | 453 | 844 | 437 | 348 | 270 | 425 | 455 | 767 | 914 | 206 |
| w | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| x | 129 | 175 | 42 | 138 | 83 | 151 | 272 | 135 | 119 | 65 | 114 | 149 | 288 | 283 | 63 |
| y | 94 | 89 | 31 | 81 | 67 | 80 | 148 | 64 | 58 | 61 | 61 | 98 | 119 | 145 | 36 |
| z | 20 | 23 | 7 | 27 | 18 | 39 | 71 | 30 | 20 | 11 | 25 | 37 | 41 | 41 | 3 |
| à | 128 | 136 | 39 | 110 | 90 | 131 | 246 | 130 | 90 | 73 | 101 | 129 | 209 | 224 | 48 |
| â | 36 | 50 | 9 | 43 | 67 | 42 | 50 | 43 | 24 | 18 | 40 | 33 | 55 | 75 | 20 |
| æ | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 2 | 0 | 0 | 0 | 3 | 0 | 2 |
| ç | 35 | 28 | 10 | 22 | 24 | 30 | 46 | 34 | 16 | 16 | 34 | 23 | 61 | 56 | 17 |
| è | 102 | 147 | 49 | 138 | 112 | 122 | 232 | 119 | 99 | 68 | 108 | 151 | 237 | 260 | 58 |
| é | 423 | 513 | 194 | 424 | 367 | 548 | 966 | 502 | 370 | 304 | 438 | 480 | 940 | 1019 | 221 |
| ê | 43 | 68 | 24 | 36 | 44 | 57 | 96 | 54 | 43 | 53 | 68 | 60 | 126 | 94 | 32 |
| ë | 1 | 0 | 0 | 0 | 1 | 0 | 2 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| î | 17 | 20 | 12 | 15 | 11 | 15 | 42 | 11 | 8 | 15 | 26 | 13 | 32 | 28 | 12 |
| ï | 2 | 0 | 0 | 2 | 8 | 12 | 9 | 1 | 2 | 5 | 15 | 3 | 5 | 2 | 0 |
| ô | 20 | 20 | 27 | 15 | 23 | 15 | 41 | 14 | 13 | 38 | 50 | 15 | 37 | 45 | 24 |
| ù | 14 | 9 | 4 | 6 | 18 | 14 | 30 | 6 | 5 | 3 | 7 | 11 | 24 | 21 | 7 |
| û | 7 | 9 | 7 | 4 | 15 | 15 | 38 | 8 | 15 | 10 | 9 | 14 | 30 | 21 | 11 |
| œ | 5 | 5 | 2 | 8 | 7 | 9 | 9 | 5 | 3 | 5 | 7 | 0 | 13 | 12 | 6 |

| | English | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 |
| a | 2217 | 2761 | 990 | 2274 | 1865 | 2606 | 4805 | 2396 | 1993 | 1627 | 2375 | 2560 |
| b | 451 | 551 | 183 | 454 | 400 | 518 | 913 | 431 | 408 | 359 | 437 | 489 |
| c | 729 | 777 | 271 | 736 | 553 | 797 | 1521 | 702 | 653 | 451 | 643 | 757 |
| d | 1316 | 1548 | 557 | 1315 | 1135 | 1509 | 2681 | 1416 | 1096 | 933 | 1364 | 1566 |
| e | 3967 | 4543 | 1570 | 3814 | 3210 | 4237 | 7834 | 4014 | 3373 | 2690 | 3790 | 4331 |
| f | 596 | 685 | 279 | 595 | 515 | 687 | 1366 | 621 | 575 | 477 | 610 | 677 |
| g | 662 | 769 | 253 | 559 | 525 | 669 | 1163 | 624 | 517 | 409 | 644 | 650 |
| h | 2060 | 2530 | 875 | 1978 | 1693 | 2254 | 4379 | 2171 | 1766 | 1475 | 2217 | 2348 |
| i | 1823 | 2163 | 783 | 1835 | 1482 | 2097 | 3838 | 2011 | 1648 | 1196 | 1830 | 2033 |
| j | 22 | 13 | 4 | 22 | 7 | 26 | 42 | 24 | 16 | 7 | 16 | 28 |
| k | 200 | 284 | 82 | 198 | 153 | 216 | 416 | 216 | 146 | 131 | 217 | 234 |
| l | 1204 | 1319 | 520 | 1073 | 949 | 1239 | 2434 | 1152 | 861 | 789 | 1122 | 1102 |
| m | 656 | 829 | 333 | 690 | 571 | 763 | 1461 | 748 | 629 | 506 | 799 | 746 |
| n | 1851 | 2218 | 816 | 1771 | 1468 | 2174 | 3816 | 2085 | 1728 | 1266 | 1833 | 2125 |
| o | 1897 | 2237 | 828 | 1865 | 1586 | 2231 | 4091 | 1947 | 1698 | 1369 | 1948 | 2105 |
| p | 525 | 606 | 194 | 514 | 517 | 613 | 1040 | 527 | 442 | 325 | 486 | 581 |
| q | 19 | 21 | 13 | 33 | 17 | 25 | 39 | 33 | 20 | 23 | 23 | 32 |
| r | 1764 | 2019 | 711 | 1726 | 1357 | 1931 | 3674 | 1915 | 1561 | 1211 | 1720 | 1939 |
| s | 1942 | 2411 | 864 | 1918 | 1646 | 2192 | 4060 | 1966 | 1626 | 1344 | 1945 | 2152 |
| t | 2547 | 3083 | 1048 | 2704 | 2178 | 2955 | 5369 | 2765 | 2442 | 1759 | 2424 | 3046 |
| u | 704 | 861 | 298 | 745 | 663 | 899 | 1552 | 789 | 683 | 502 | 767 | 750 |
| v | 258 | 295 | 94 | 245 | 194 | 277 | 465 | 266 | 208 | 181 | 246 | 278 |
| w | 653 | 769 | 254 | 663 | 568 | 733 | 1332 | 695 | 560 | 410 | 632 | 721 |
| x | 29 | 37 | 8 | 60 | 26 | 49 | 74 | 65 | 25 | 31 | 20 | 35 |
| y | 401 | 475 | 145 | 467 | 330 | 464 | 843 | 379 | 328 | 255 | 457 | 418 |
| z | 18 | 31 | 15 | 19 | 33 | 37 | 52 | 24 | 18 | 20 | 39 | 40 |
| à | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| â | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| æ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ç | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| è | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| é | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ê | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ë | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| î | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ï | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ô | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ù | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| û | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| œ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table: Character counts per chapter in French, left part, and English, right part

Each characters is modeled by a vector of chapters.

# Singular Value Decomposition

There are as many as 40 characters: the 26 unaccented letters from *a* to *z* and the 14 French accented letters

**Singular value decomposition** (SVD) reduces these dimensions, while keeping the resulting vectors semantically close

$\mathbf{X}$ is the $m \times n$ matrix of the letter counts per chapter, in our case, $m = 30$ and $n = 40$.

We can rewrite $\mathbf{X}$ as:

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^\intercal,$$

where $\mathbf{U}$ is a matrix of dimensions $m \times m$, $\Sigma$, a diagonal matrix of dimensions $m \times n$, and $\mathbf{V}$, a matrix of dimensions $n \times n$.

The diagonal terms of $\Sigma$ are called the **singular values** and are traditionally arranged by decreasing value.

We keep the highest values and set the rest to zero.

# Code Example

Jupyter Notebook 3.1-SVD

# 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*, ...

# *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* |

# 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
```

# 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 $P(It\ was\ a\ bright\ cold\ day\ in\ April)$ 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:

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

# Maximum Likelihood Estimate

Bigrams:

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

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

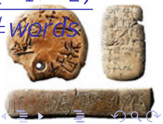
$$\frac{C(w_{i-1}, w_i)}{\#bigrams}.$$

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:

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



# Training the Model

The model is trained on a part of the corpus: the **training set**

It is tested on 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>

| $w_{i-1}, w_i$ | $C(w_{i-1}, w_i)$ | $C(w_{i-1})$ | $P_{MLE}(w_i \vert w_{i-1})$ |
|---|---|---|---|
| <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:

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

- 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

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

| $w_i, w_{i+1}$ | $C(w_i, w_{i+1}) + 1$ | $C(w_i) + Card(V)$ | $P_{Lap}(w_{i+1}\|w_i)$ |
|---|---|---|---|
| <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 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 $86352^2 = 74{,}563{,}225$ 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:

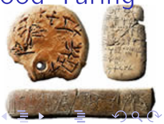
$$P_{\text{Backoff}}(w_i|w_{i-1}) = \begin{cases} \tilde{P}(w_i|w_{i-1}), & \text{if } C(w_{i-1}, w_i) \neq 0, \\ \alpha P(w_i), & \text{otherwise.} \end{cases}$$

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

# Backoff: Example

| $w_{i-1}, w_i$ | $C(w_{i-1}, w_i)$ | | $C(w_i)$ | $P_{Backoff}(w_i \mid w_{i-1})$ |
|---|---|---|---|---|
| <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:

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

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:

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

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

...

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

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

Here the lower, the better

The figures are usually presented with the perplexity metric:

$$PP(p,m) = 2^{H(L)}.$$

# Mathematical Background

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

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.

# 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
```

# Word Embeddings

We can extend singular value decomposition from characters to words.
The rows will represent the words in the corpus, and the columns, documents,
We can replace documents by a context of a few words to the left and to the right of the focus word: $w_i$.
A context $C_j$ is then defined by a window of $2K$ words centered on the word:

$$w_{i-K}, w_{i-K+1}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+K-1}, w_{i+K},$$

where the context representation uses a bag of words.
We can even reduce the context to a single word to the left or to the right of $w_i$ and use bigrams.

# Word Embeddings

We store the word-context pairs $(w_i, C_j)$ in a matrix.
Each matrix element measures the association strength between word $w_i$ and context $C_j$, for instance mutual information.
Mutual information, often called pointwise mutual information (the strength of an association) is defined as:

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

| D#\Words | $C_1$ | $C_2$ | $C_3$ | ... | $C_n$ |
|---|---|---|---|---|---|
| $w_1$ | $MI(w_1, C_1)$ | $MI(w_1, C_2)$ | $MI(w_1, C_3)$ | ... | $MI(w_1, C_n)$ |
| $w_2$ | $MI(w_2, C_1)$ | $MI(w_2, C_2)$ | $MI(w_2, C_3)$ | ... | $MI(w_2, C_n)$ |
| $w_3$ | $MI(w_3, C_1)$ | $MI(w_3, C_2)$ | $MI(w_3, C_3)$ | ... | $MI(w_3, C_n)$ |
| ... | ... | ... | ... | ... | ... |
| $w_m$ | $MI(w_m, C_1)$ | $MI(w_m, C_2)$ | $MI(w_m, C_3)$ | ... | $MI(w_m, C_n)$ |

# Word Embeddings

We compute the word embeddings with a singular value decomposition, where we truncate the matrix $\mathbf{U}\Sigma$ to 50, 100, 300, or 500 dimensions.
The word embeddings are the rows of this matrix.
We usually measure the similarity between two embeddings $\mathbf{u}$ and $\mathbf{v}$ with the cosine similarity:

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{||\mathbf{u}|| \cdot ||\mathbf{v}||},$$

ranging from -1 (most dissimilar) to 1 (most similar) or with the cosine distance ranging from 0 (closest) to 2 (most distant):

$$1 - \cos(\mathbf{u}, \mathbf{v}) = 1 - \frac{\mathbf{u} \cdot \mathbf{v}}{||\mathbf{u}|| \cdot ||\mathbf{v}||}.$$

# Popular Word Embeddings

Embeddings from large corpora are obtained with iterative techniques

Some popular embedding algorithms with open source programs:

- word2vec: `https://github.com/tmikolov/word2vec`
- GloVe: Global Vectors for Word Representation `https://nlp.stanford.edu/projects/glove/`
- ELMo: `https://allennlp.org/elmo`
- fastText: `https://fasttext.cc/`

To derive word embeddings, you will have to apply these programs on a very large corpus

Embeddings for many languages are also publicly available. You just download them

gensim is a Python library to create word embeddings from a corpus. `https://radimrehurek.com/gensim/index.html`