Text Preprocessing

CHIEN CHIN CHEN

Outline

Tokenization

Stopwords

Stemming and Lemmatization

Tokenization of Mandrian

Zipf’s law

Preface

To take advantage of tools of *machine learning* and *deep learning*, we need to extract **structured numerical data** (vectors) from **natural language**!!

The conversion of text data to numerical vectors is called text **PRE-PREOCESSING**.

- Involve several steps: **tokenization**, **stemming**, **stopword removal**, **term weighting**
- Not all steps are necessary (e.g., stopwrod removal)

In this topic, we learn concepts and demonstrate **codes** for text pre-processing.

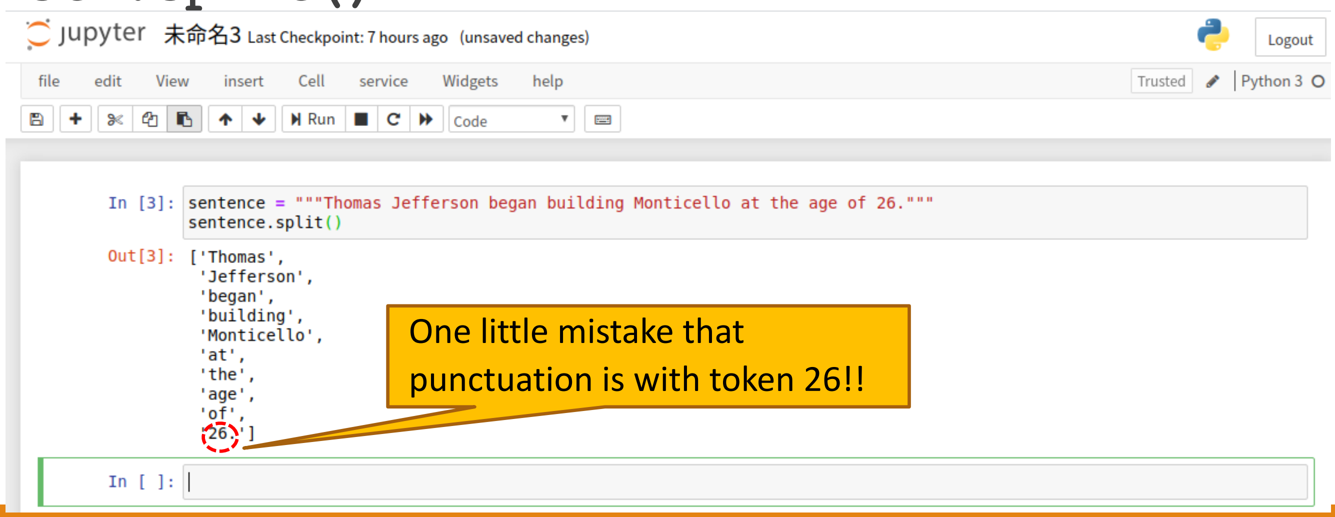
Tokenization (1/5)

Tokenization is usually the <u>first step</u> in an NLP system.

- It breaks unstructured text into chunks of **tokens** (words).

A simple way to tokenize a text is to use “**whitespace**” within a string as the delimiter of tokens.

- It is easy if you are familiar with Python built-in method **`str.split()`**



Tokenization (2/5)

Basically, extracting meaningful (correct) tokens is difficult, and **no universe solution exists**.

- "Best day everrrrrrrr" "Awesommmmmmmmeeeeeeeee day :)"
- "Don't do that" → ['Don't', 'do', 'that'] or ['Do', 'not', 'do', 'that']
- "ice cream" → ['ice', 'cream'] or ['ice cream']

Later, we show several **language packages** to help us construct good quality tokenization.

But now … with a bit more Python, you can create a vector representation for a word; and further represent a text as a sequence of vectors.

- The vector of a word is also called **one-hot vector**.

Tokenization (3/5)

One-hot vector: all but **one** of the positions in a vector are 0.

```
In [5]: import numpy as np
        token_sequence = str.split(sentence)
        vocab = sorted(set(token_sequence))

In [6]: vocab

Out[6]: ['26.',
         'Jefferson',
         'Monticello',
         'Thomas',
         'age',
         'at',
         'began',
         'building',
         'of',
         'the']

In [7]: num_tokens = len(token_sequence)
        vocab_size = len(vocab)

In [10]: print(num_tokens, vocab_size)

10 10

In [11]: onehot_vectors = np.zeros((num_tokens, vocab_size), int)

In [12]: for i, word in enumerate(token_sequence):
             onehot_vectors[i, vocab.index(word)] = 1

In [13]: onehot_vectors

Out[13]: array([[0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
                [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
                [0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
                [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
                [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]])

In [ ]:
```

One-hot vector of the first token

tokens

```
array([[0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
       [0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
       [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
```

vocabulary: the set of unique tokens in the text

Tokenization (4/5)

Congrats!! You've turned a natural language sentence into <u>a sequence of vectors</u>.

Computer systems (learning algorithms) have a chance to read and do math on the vectors to accomplish your NLP works ☺

- Actually, text's one-hot vectors are typically used in neural nets, especially sequence-to-sequence models.

But … not so fast!! Let's polish the tokens that could further enhance the performance of text mining models.

Tokenization (5/5)

Many Python libraries implement word tokenizer.

- **keras**.preprocessing.text.Tokenizer, **nltk**.tokenize.TreebankWordTokenizer, **nltk**.tokenize.word_tokenize

Let's practice with `word_tokenize`

```
In [1]: from nltk.tokenize import word_tokenize

In [2]: source = "Hi, your phone doesn't work but mine is okay; it is so weird. Bye."

In [3]: source
Out[3]: "Hi, your phone doesn't work but mine is okay; it is so weird. Bye."

In [4]: tokens_byWordTokenize = word_tokenize(source)

In [5]: tokens_byWordTokenize
Out[5]: ['Hi',
 ',',
 'your',
 'phone',
 'does',
 "n't",
 'work',
 'but',
 'mine',
 'is',
 'okay',
 ';',
 'it',
 'is',
 'so',
 'weird',
 '.',
 'Bye',
 '.']
```

So good,
it tokenizes
"**doesn't**" into "**does**" and "**n't**"

But punctuations (e.g., ". , ;")
are still there…

What if you wanna remove those punctuations?

```
In [6]: stop_punc = [',',';','.']

In [7]: final_tokens = [x for x in tokens_byWordTokenize if x not in stop_punc]

In [8]: final_tokens
Out[8]: ['Hi',
 'your',
 'phone',
 'does',
 "n't",
 'work',
 'but',
 'mine',
 'is',
 'okay',
 'it',
 'is',
 'so',
 'weird',
 'Bye']
```

Just compile a list

Some tokenizers output a separate token for a sentence-ending punctuation (e.g., . ; ? !) because those tokens could be helpful for a NLP task.

Tokenize Text from Social Media (1/2)

Texts from social networks are difficult to deal with

- They generally contain a lot of informal words
  - E.g., Best Day Everrrrrrrrrr :)
- But in many business applications, social texts are very informative!!

Do not worry, here comes tools to help us – `casual_tokenize` and `TweetTokenizer`

Tokenize Text from Social Media (2/2)

```
In [1]: from nltk.tokenize.casual import casual_tokenize

In [2]: source = "Donald J. Trump @realDonaldTrump Best day everrrrr. Toooo Awesomemmmmeeeeee :*)  <3 :)"

In [3]: casual_tokenize(source)
Out[3]: ['Donald',
 'J',
 '.',
 'Trump',
 '@realDonaldTrump',
 'Best',
 'day',
 'everrrrr',
 '.',
 'Toooo',
 'Awesomemmmmeeeeee',
 ':*)',
 '<3',
 ':)']

In [4]: casual_tokenize(source, strip_handles=True, reduce_len=True)
Out[4]: ['Donald',
 'J',
 '.',
 'Trump',
 'Best',
 'day',
 'everrr',
 '.',
 'Tooo',
 'Awesomemmmeee',
 ':*)',
 '<3',
 ':)']
```

Replace repeated character sequences of length 3 or greater with sequences of length 3.

```
from nltk.tokenize import word_tokenize

word_tokenize(source)

['Donald',
 'J.',
 'Trump',
 '@',
 'realDonaldTrump',
 'Best',
 'day',
 'everrrrr',
 '.',
 'Toooo',
 'Awesomemmmmeeeeee',
 ':',
 '*',
 ')',
 '<',
 '3',
 ':',
 ')']
```

Stop Words (1/2)

Stop words are common (function) words that occur with a high frequency but carry less information.

- Examples: a, an, the, this, of, …

Stop words are supposed to be excluded from NLP pipeline.

- But, they sometimes help provide important information…
  - Mark reported to the CEO … → Mark reported CEO …
  - Mark reported as the CEO … → Mark reported CEO …

Many term weighting schemes help determine the **WEIGHT** of terms (tokens).

- So, in some applications, all tokens are reserved for text mining.

Stop Words (2/2)

Multiple English stopword lists are available in the Internet.

- E.g., `nltk` and `scikit-learn`
- Let's try `nltk` stopword list.

```
In [1]: from nltk.tokenize import word_tokenize

In [2]: source = "Your phones are not working but mine is okay."

In [3]: tokens_byWordTokenize = word_tokenize(source)

In [4]: stop_punc  = [ ',' , ';' , '.' ]
        tokens_byWordTokenize = [x for x in tokens_byWordTokenize if x not in stop_punc]
        print(tokens_byWordTokenize)

        ['Your', 'phones', 'are', 'not', 'working', 'but', 'mine', 'is', 'okay']

In [5]: import  nltk
        nltk.download('stopwords')
        stop_words = nltk.corpus.stopwords.words('english')
        len ( stop_words )

        [nltk_data] Downloading package stopwords to /home/paton/nltk_data...
        [nltk_data]   Package stopwords is already up-to-date!

Out[5]: 179

In [6]: stop_words[:7]

Out[6]: ['i', 'me', 'my', 'myself', 'we', 'our', 'ours']

In [7]: tokens_without_stopwords = [x for x in tokens_byWordTokenize if x not in stop_words]

In [8]: print(tokens_without_stopwords)

        ['Your', 'phones', 'working', 'mine', 'okay']
```

Hmm…we have removed 'not'…

Normalization

Normalization: tokens that mean similar things are combined into a single, normalized form.

- *Good* vs *good*
- *Operates*, *operated*, *operating*, and *operation*.

Doing normalization helps reduce the size of your vocabulary and improve the association of different spellings of a token.

Frequently-used normalization procedures:

- Case folding
- Stemming
- Lemmatization

Case Folding (1/2)

English words are capitalized because of their presence at the beginning of a sentence.

The simplest way to normalize the case of a text string is to lowercase all the characters.

- Python built-in `str.lower()` function.

But this will also normalize away some meaningful capitalization.

- FedEx → fedex.

A better approach is to lowercase only the first word of a sentence.

- Is complicated and may still introduce errors for proper nouns that start a sentence.
- “Joy is filled with joy.”

Case Folding (2/2)

Again, **no universe solution for text preprocessing**, some NLP systems even do not normalize for case at all.

But note that case normalization is particularly useful for **search engines**.

- Because users are so lazy …
- When searching for "*iphone*" you would get information about "*iPhone*"

Stemming (1/4)

Stemming: Remove **suffixes** from words to combine words with similar meanings together under their common **stem**.

- **A stem is not required to be a properly spelled word**!!

Most stemming methods are rule-based. One may define a rule *'ing'* → *''* to remove suffix '*ing*'

- *ending* → *end* (good)
- *running* → *runn* (bad, because we cannot group running, run, and runs together)
- *sing* → *s* (so bad ☹)

Stemming (2/4)

Never ever think rule defining is easy!!

Fortunately, we can make use of stemming packages.

- encourage you to do so ☺

The well-known **Porter stemmer**, named for the computer scientist **Martin Porter**.

His 1980 paper "An algorithm for suffix stripping", proposing the stemming algorithm, has been cited over 8000 times. (https://en.wikipedia.org/wiki/Martin_Porter)

You can admire the 300 lines of code @ https://github.com/jedijulia/porter-stemmer/blob/master/stemmer.py that Mr. Porter put **his lifetime of refinement on them** ☺

Stemming (3/4)

It consists of eight steps (1a, 1b, 1c, 2, 3, 4, 5a, and 5b)

- We are not going to the detail of those steps and rules.
- See the stems below; many of them are not correct words!!

```
In [37]: stemmer.stem("operate")
Out[37]: 'oper'

In [38]: stemmer.stem("operating")
Out[38]: 'oper'

In [39]: stemmer.stem("operates")
Out[39]: 'oper'

In [5]: stemmer.stem('cement')
Out[5]: 'cement'
```

Stemming (4/4)

Let’s refine the previous example: ‘*Your phones are not working but mine is okay.*’

```
In [8]:  from nltk.stem.porter import PorterStemmer

In [9]:  stemmer = PorterStemmer()

In [10]: stemmed_tokens = [stemmer.stem(x) for x in tokens_without_stopwords]

In [11]: print(stemmed_tokens)

         ['your', 'phone', 'work', 'mine', 'okay']

In [12]: print(tokens_without_stopwords)

         ['Your', 'phones', 'working', 'mine', 'okay']
```

Lemmatization (1/2)

Lemmatization Is a **more accurate** way to normalize a word.

- To output the **lemma** of a word (the root word).

Lemmatizer generally uses a knowledge base (e.g., WordNet) to identify the lemma of a word.

- Also, a word's **part-of-speech** (POS) is used to ensure a correct output.

WordNet:

PRINCETON UNIVERSITY

WordNet

A Lexical Database for English

About WordNet

WordNet® is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations. The resulting network of meaningfully related words and concepts can be navigated with the browser. WordNet is also freely and publicly available for download. WordNet's structure makes it a useful tool for computational linguistics and natural language processing.

```
In [16]: lemmatizer.lemmatize("ate", pos="v")
Out[16]: 'eat'
In [14]: stemmer.stem('ate')
Out[14]: 'ate'
```

pos: default is noun

```
In [21]: lemmatizer.lemmatize("are", pos="v")
Out[21]: 'be'

In [22]: lemmatizer.lemmatize("were", pos="v")
Out[22]: 'be'
```

Lemmatization (2/2)

Let's practice
`WordNetLemmatizer`

```
In [1]: import nltk

In [2]: nltk.download('wordnet')

        [nltk_data] Downloading package wordnet to /home/paton/nltk_data...
        [nltk_data]   Package wordnet is already up-to-date!

Out[2]: True

In [3]: from nltk.stem import WordNetLemmatizer
        lemmatizer = WordNetLemmatizer()

In [4]: lemmatizer.lemmatize("operating", pos="v")

Out[4]: 'operate'

In [5]: lemmatizer.lemmatize("operate", pos="v")

Out[5]: 'operate'

In [6]: lemmatizer.lemmatize("operates", pos="v")

Out[6]: 'operate'
```

Note that NLTK `WordNetLemmatizer` is restricted to the Princeton WordNet.

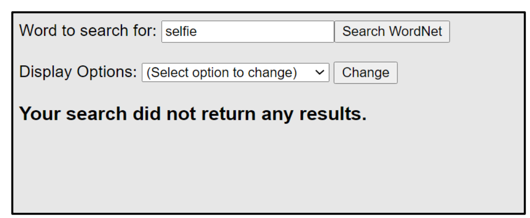
- The knowledge is huge and accurate … but not complete; cannot handle Internet slangs!!
- So, sometimes, the lemmatized tokens may not be perfect.

```
In [32]: lemmatizer.lemmatize('selfie', pos="n")

Out[32]: 'selfie'

In [31]: lemmatizer.lemmatize('selfies', pos='n')

Out[31]: selfies
```



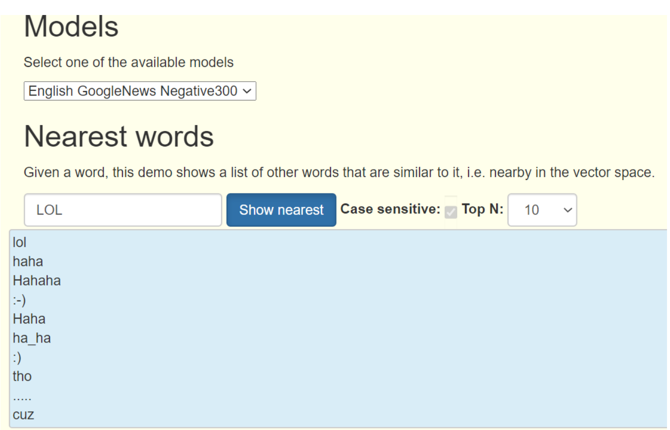
Stemming and Lemmatization

Stemming and lemmatization are popular in traditional text mining.

- Some even suggest using a lemmatizer right before a stemmer.
  - Try best to identify valid English words first, then do stemming on them.

But many deep learning approaches do not have to use them!!

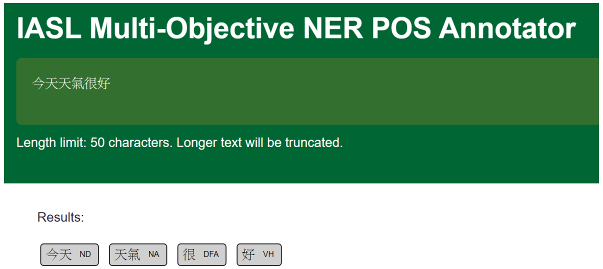
- Some neural net techniques making words with similar meaning closer
  - E.g., word embedding



Tokenization of Mandarin

Many tools for Mandarin word segmentation

jieba



http://monpa.iis.sinica.edu.tw:9000/chunk

I fully support CKIP … but now `CkipTagger` can only be installed under tensorflow 1.13.1 ~ 2

- https://github.com/ckiplab/ckiptagger

Jieba (1/3)

See https://github.com/fxsjy/jieba to install jieba in your environment

- `pip install jieba`

```
In [1]: import jieba

In [2]: sentence = "今天天氣很不錯,管中閔校長邀大家去陽明山爬山."

In [3]: print(sentence)

        今天天氣很不錯,管中閔校長邀大家去陽明山爬山.

In [4]: result = jieba.cut(sentence)

In [5]: tokens= list(result)

        Building prefix dict from the default dictionary ...
        Loading model from cache /tmp/jieba.cache
        Loading model cost 1.566 seconds.
        Prefix dict has been built successfully.

In [6]: tokens

Out[6]: ['今天', '天氣', '很', '不錯', ',', '管中', '閔校', '長', '邀', '大家', '去陽', '明山', '爬山', '.']
```

Jieba (2/3)

You can enhance the output by using a traditional Chinese dictionary

- https://github.com/fxsjy/jieba/raw/master/extra_dict/dict.txt.big
- Save it as a local file, then load it

```
In [7]: jieba.set_dictionary('./KerasExamples/dict.txt.big')

In [8]: new_result = jieba.cut(sentence)

In [9]: new_tokens = list(new_result)

        Building prefix dict from /home/paton/KerasExamples/dict.txt.big ...
        Loading model from cache /tmp/jieba.ude4ee7469a5643dfc8281e421fdfb29c.cache
        Loading model cost 2.609 seconds.
        Prefix dict has been built successfully.

In [10]: new_tokens

Out[10]: ['今天天氣', '很', '不錯', ',', '管中', '閔', '校長', '邀', '大家', '去', '陽明山', '爬山', '.']
```

Jieba (3/3)

You can include your own dictionary in addition to the jieba dictionary.

- To help tokenize new words (people names, proper nouns).

```
In [11]: jieba.load_userdict('./KerasExamples/my.dict.txt')

In [12]: final_result = jieba.cut(sentence)

In [13]: final_toknes = list(final_result)

In [14]: final_toknes
Out[14]: ['今天天氣', '很', '不錯', ',', '管中閔', '校長', '邀', '大家', '去', '陽明山', '爬山', '.']
```

```
檔案(F) 編輯(E) 檢視(V) 搜尋(S) 終端機(T) 求助(H)
(base) paton@paton-VirtualBox:~$ more ./KerasExamples/my.dict.txt
管中閔 1 n
(base) paton@paton-VirtualBox:~$
```

Mandarin Stop Words

Do we have Mandarin stop words?

- http://www.ranks.nl/stopwords/chinese-stopwords
- https://github.com/fxsjy/jieba/blob/master/extra_dict/stop_words.txt
- http://www.aclclp.org.tw/doc/wlawf_abstract.pdf

中央研究院平衡語料庫詞集及詞頻統計

詞項	詞類	頻率	累積頻率
的	DE	285826	5.82
是	SHI	84014	7.53
一	Neu	58388	8.72
在	P	56769	9.88
有	V_2	45823	10.81
個	Nf	41077	11.64
我	Nh	40332	12.47
不	D	39014	13.26
這	Nep	33659	13.95
了	Di	31873	14.59

Zipf’s Law

1930s, the American linguist *George Kingsley Zipf* formulate the relation of term frequency and rank.

For a sufficiently **large corpus**, the frequency of any word is inversely proportional to its rank in the frequency table.

- The first term in the ranked list will appear twice as often as the second
- And three times as often as the third …

Not jut words, Zipf’s Law applies to a lot of counting.

- E.g., city population.

George Kingsley Zipf

1917 photograph from the 1919 Annual of the Freeport High School, Freeport, Illinois

Born	January 7, 1902 Freeport, Illinois
Died	September 25, 1950 (aged 48) Newton, Massachusetts
Nationality	American
Alma mater	Harvard College
Known for	Zipf's law
Spouse(s)	Joyce Waters Brown Zipf
Children	Robert Zipf, Katherine Sandstrom, Joyce Harrington, Henry Zipf
	Scientific career
Fields	Statistics, linguistics

Take Brown Corpus as an Example

The **Brown Corpus** was the first million-word electronic corpus of English, created in 1961 at Brown University.

- Over the following several years part-of-speech tags were applied. The Greene and Rubin tagging program helped in this, with extensive manual proofreading.

Automatic Grammatical Tagging of English

Barbara B. Greene, Gerald M. Rubin
Department of Linguistics, Brown University, 1971 - 306 頁
0 書評

```
In [1]: import nltk
        nltk.download('brown')

[nltk_data] Downloading package brown to /home/paton/nltk_data...
[nltk_data]   Unzipping corpora/brown.zip.

Out[1]: True

In [2]: from nltk.corpus import brown
        brown.words()[:5]

Out[2]: ['The', 'Fulton', 'County', 'Grand', 'Jury']
```

```
In [6]: from collections import Counter
        puncs = set((',', '.', '--', '-', '!', '?', ':', ';', '``', "''", '(', ')', '[', ']'))
        word_list = (x.lower() for x in brown.words() if x not in puncs)
        token_counts = Counter(word_list)

In [7]: token_counts.most_common(20)

Out[7]: [('the', 69971),
         ('of', 36412),
         ('and', 28853),
         ('to', 26158),
         ('a', 23195),
         ('in', 21337),
         ('that', 10594),
         ('is', 10109),
         ('was', 9815),
         ('he', 9548),
         ('for', 9489),
         ('it', 8760),
         ('with', 7289),
         ('as', 7253),
         ('his', 6996),
         ('on', 6741),
         ('be', 6377),
         ('at', 5372),
         ('by', 5306),
         ('i', 5164)]
```

"the" occurs roughly twice as often as "of", and roughly three times as often as "and"

So sad ... top frequent words are all stop words!!

More on Zipf's Law (1/2)

If we rank terms according to their **collection frequency**

- Collection frequency: the number of times a term appears in a text collection. Then the collection frequency cf_i of the *i*th most common term is proportional to $1/i$.

$$cf_i \propto \frac{1}{i} \quad \text{or} \quad cf_i \cdot i = c$$

a constant

- Equivalently, we can write Zipf's Law as

$$cf_i = ci^k$$

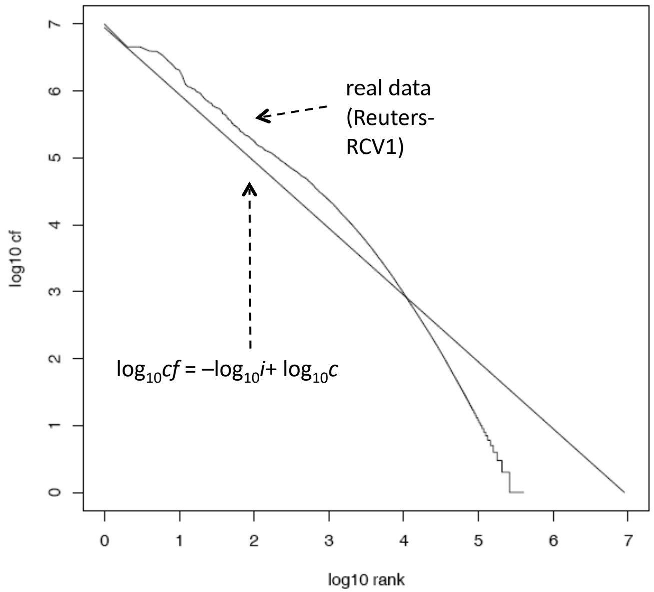
$k = -1$

and

$$\log cf_i = \log ci^{-1}$$

$$= -\log i + \log c$$

Rank and collection frequency is linear in log-log space.



More on Zipf's Law (2/2)

Zipf's Law implies that …

- **There are a few very common words** (e.g., 'the', 'of', 'and' …)
- **And many low frequency words.**