Miscalleneous

- Hiwi-Job-Angebot DFKI (siehe read.mi)
- Hinweis: QIS-Anleitung (siehe read.mi)
- Slides letztes Kapitel (Feedback) nachholen

Applications of Artificial Intelligence

– Winter Term 21/22 –

Chapter 04

Natural Language Processing

Prof. Dr. Adrian Ulges

RheinMain University of Applied Sciences

Outline

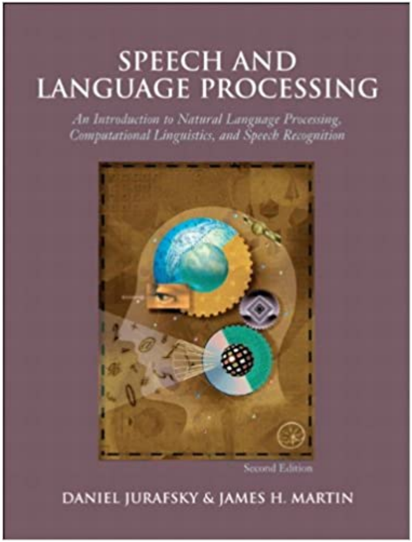
1. Fundamentals

2. Preprocessing and Heuristics

3. Statistical Word Similarity

4. Syntactical Word Similarity

Recommended Read [3]



Natural Language Processing: Some Tasks[1]

Part-of-Speech Tagging

Abraham Lincoln was the 16th President of the United States. He was shot by stage actor John Wilkes Booth. It was in his interest ...

shot → VBN *(verb past partiple)*

interest → NN *(noun singular)*

Coreference Resolution

Abraham Lincoln was the 16th President of the United States. He was assassinated by stage actor John Wilkes Booth on April 14, 1865.

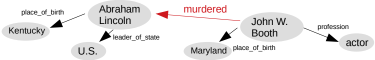
Named Entity Recognition

Abraham Lincoln was the 16th President of the United States. He was assassinated by stage actor John Wilkes Booth on April 14, 1865.

COUNTRY PERSON DATE

Information Extraction

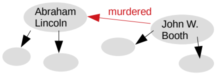
Abraham Lincoln was the 16th President of the United States. He was assassinated by stage actor John Wilkes Booth on April 14, 1865.



Question Answering

> "Who killed Abe Lincoln?"

Abraham Lincoln was the 16th President of the United States. He was assassinated by stage actor John Wilkes Booth on April 14, 1865.



Goal-oriented Dialog Understanding

user

> What's a good diner nearby?

> How far is this?

bot

> How about the Taj Mahal?

> 500 m

> "How about Italian ones?"

restaurant	groceries	travel plan	travel plan	
japanese	italian	french	ethiopian	turkish

[1] The Natural Language Decathlon: https://github.com/salesforce/decaNLP

Natural Language Processing: Basics

Morphology

- analyzes the structure and parts of words (`walk - ed`).

Syntax *(greek "order together")*

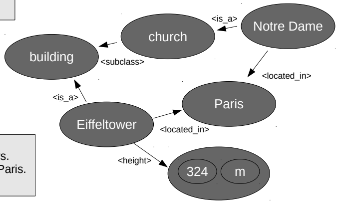
- How are words ordered? What is a sentence's structure?

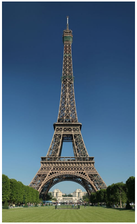
Semantics

- What is the **meaning** of a word/sentence?
- **Goal** of many NLP techniques
  - **classification** of sentences / questions
  - **extraction** of entities / facts

Text → Semantics?

Is the Eiffeltower higher than Notre Dame ?





The Eiffeltower is an iron lattice tower on the Champ de Mars. With its **height of 324 meters**, **it** is the **tallest structure** in Paris.

Transferring natural language into a formal description is **not entirely solved** so far.

Example: The Research Project **Read The Web**[2]

cracking_new_year is a monarch	1045	30-mar-2017	96.3
start_search_field is a research project	1046	02-apr-2017	98.8
adrianne_curry is a fashion model	1045	30-mar-2017	99.8

tuna is a type of large_predatory_fish	1046	02-apr-2017
tampa_bay_devil_rays is headquartered in the state or province new_york	1050	19-apr-2017
man is an animal that can develop disability	1046	02-apr-2017

[2] http://rtw.ml.cmu.edu/rtw/

Outline

Text Preprocessing

Often, the first step in NLP systems is preprocessing text.
Here, two possible steps are:

1. Segmentation
   - Segmenting text into **sentences**
   - Segmenting sentences intox **words/terms/tokens** *("house", "rock 'n roll", "#nlproc", "getStringFromObject()"?)*
   - This is often done using **regular expressions**.

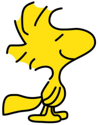
2. Normalization
   - reducing words to their base form *(aka 'lemma')*.
   - e.g., lemmatization, stemming, compound splitting.

Text Preprocessing (cont'd)

Documents	Tokens (preprocessed)	Vocabulary
"The two most important days in your life are the day you are born and the day you find out why."	{ '**day**', 'most', 'two', 'born', 'in', 'out', 'why', 'are', 'you', 'life', 'import', 'the', 'find', 'and' }	{ 'enjoy', 'to', 'you', 'so', 'out', 'import', 'the', 'why', 'day', 'most', 'and', 'everi', 'of', 'is', 'two', 'born', 'in', 'learn', 'are', 'life', 'second', 'find' }
"Every second of your life is important so learn to enjoy"	{ '**everi**', 'of', 'is', 'import', 'enjoy', 'learn', 'to', 'you', 'life', 'so', 'second' }	

3. Vocabulary
   - Collect all tokens from the corpus in a vocabulary $\{t_1, \ldots, t_m\}$.
   - Usually, we filter tokens that are too frequent, too infrequent, numbers, special tokens, etc.

Regular Expressions



- … match patterns in strings, using a compact formal language.
- Symbols in []-brackets: logical OR.

[Ww]ood	Wood / wood	**Wood**stock is small indeed.
[0 – 9] or \d	some digit	Beverly Hills **9**0210.
yours|mine	alternative: pipe symbol	Brace **yours**elf.
[a-z]	Kleinbuchstaben	W**o**odstock is small indeed.

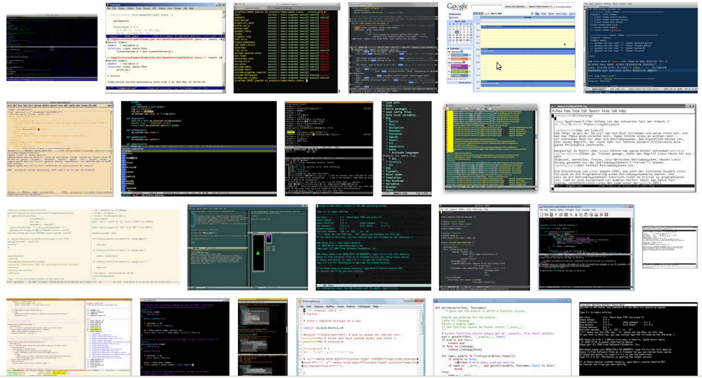
- Negation: [ˆX] means “not X”.

[ˆA-Z]	anything but capital letters	W**o**odstock.
[ˆsS]	anything but s or S	S**u**re you can.
aˆb	aˆb (no negation, since not in brackets)	Sure you can.

- Other Options:

colou?r	u is optional	**color**s and **colours**.
ye*ah	0 or more occurrences	**yah**! **yeah**! **yeeeah**!
ye+ah	1 or more occurrences	yah! **yeah**! **yeeeah**!
beg.n	any symbol except \n	begn **begin began beg3n**
.*	any sequence of symbols	**abcdefghijklmnop**
yeah$	yeah at end of string	yeah yeah **yeah**

Regular Expressions in Python



Applications (e.g. in QA)

- **Segmenting**, rule-based **recognition** of dates, prices, ...
- **Features** for QA's machine learning components *(found abbreviation? → maybe question of type "definition")*.

Rule-based Inference with RegExp: Hearst Patterns [1]

Pattern		Example
NP such as {**NP**,}* {(and/or)} **NP**	→	**Football teams**, such as **the Eagles** or the **Patriots**, ...
NP, **especially** {NP,}* {and/or} **NP**	→	I really like **animals**, especially **birds of prey**.

- Lexico-syntactic patterns to recognize hyponymy.
- <NP>s are so-called **noun phrases**, such as *birds of prey* or *the Eagles (more later)*.
- Typical for pattern-based approaches:
  high precision ☺, low recall ☹.
- Some Patterns in Python [3] (47 patterns total) are below.

```
[
  '(<NP>\\w+ (, )?such as (<NP>\\w+ ?(, )?(and / or )?)+)',
  '(such <NP>\\w+ (, )?as (<NP>\\w+ ?(, )? (and / or )?)+)',
  '((<NP>\\w+ ?(, )?)+(and / or )? other <NP>\\w+)'
]
```

[3] https://github.com/mmichelsonIF/hearst_patterns_python

Words: Reduction to Base Forms

- Words are the smallest **meaningful units** of a language.
- Words come in a **baseform** and multiple **variants**.
- Often, we **normalize/canonicalize** words to their base form.

Canonicalization

- **from flexion form** to base form

Hauses → *Haus*
went → *go*

- **derivational forms** to base form

Bücherei → *Buch*

- from **compounds** into components *(especially for German)*

Haustür → *Haus* + *Tür*
Osterei → *Ostern* + *Ei*
Malerei $\not\to$ *Maler* + *Ei*

Words: Reduction to Base Forms

Lemmatization

- flexion form → base form *(above, e.g.* `housing` → `house`*).*
- A simple form of lemmatization is **stemming**: simply truncate a word suffix (`housing` → `hous`).

Stemming: Approaches

1. **Lexical Methods**
   - store base form in dictionaries
   - creation and maintenance of dictionary costly ☹
   - interesting for languages with **strong flexion** *(e.g., German)*
2. **Rule-based Methods**
   - use rules to change / remove suffixes
   - example: **Porter Stemmer** for English [3]

```
ing → - :  walking → walk
ies → i :  ponies → poni
```

Parts of Speech *(dt. Wortarten)*

The four most important **“open” parts of speech**.

Nouns *(dt. Nomen)*

- … refer to concrete things (`ship`), abstract things (`bandwidth`), actions (`belief`), events (`inspection`), …
- … can be accompanied by determiners (<u>`some`</u> `bandwidth`) und possessives (<u>`his`</u> `ship`).
- special case: **proper nouns** *(dt. Eigennamen)*, e.g. `Colorado`.

Verbs

- … refer to **activities** and **actions**
- … are changed through **flexion**: `eat`, `eats`, `ate`, `eaten`

Adjectives

- … refer to **properties** of **nouns** (<u>`red`</u> `ship`)

Adverbs

- … refer to **properties** of **verbs** (`he walked` <u>`slowly`</u>)

How many Parts of Speech are there in English?

45 *(in the famous “Penn Treebank”)*

Tag	Description	Example	Tag	Description	Example
CC	coordin. conjunction	*and, but, or*	SYM	symbol	+,%, &
CD	cardinal number	*one, two*	TO	“to”	*to*
DT	determiner	*a, the*	UH	interjection	*ah, oops*
EX	existential ‘there’	*there*	VB	verb base form	*eat*
FW	foreign word	*mea culpa*	VBD	verb past tense	*ate*
IN	preposition/sub-conj	*of, in, by*	VBG	verb gerund	*eating*
JJ	adjective	*yellow*	VBN	verb past participle	*eaten*
JJR	adj., comparative	*bigger*	VBP	verb non-3sg pres	*eat*
JJS	adj., superlative	*wildest*	VBZ	verb 3sg pres	*eats*
LS	list item marker	*1, 2, One*	WDT	wh-determiner	*which, that*
MD	modal	*can, should*	WP	wh-pronoun	*what, who*
NN	noun, sing. or mass	*llama*	WP$	possessive wh-	*whose*
NNS	noun, plural	*llamas*	WRB	wh-adverb	*how, where*
NNP	proper noun, sing.	*IBM*	$	dollar sign	$
NNPS	proper noun, plural	*Carolinas*	#	pound sign	#
PDT	predeterminer	*all, both*	“	left quote	‘ or “
POS	possessive ending	*’s*	”	right quote	’ or ”
PRP	personal pronoun	*I, you, he*	(	left parenthesis	[, (, {, <
PRP$	possessive pronoun	*your, one’s*	)	right parenthesis	],), }, >
RB	adverb	*quickly, never*	,	comma	,
RBR	adverb, comparative	*faster*	.	sentence-final punc	. ! ?
RBS	adverb, superlative	*fastest*	:	mid-sentence punc	: ; ... – -
RP	particle	*up, off*			

Part-of-Speech (POS) Tagging

Parts of Speech (at least in English) are (highly) ambiguous!

`book` that flight.	*vs.*	hand me this `book`.
book `that` flight.	*vs.*	I know `that` you stink.

- Most words (86%) have only one part of speech, but some **frequent** words have **multiple** POSs (`that`, `back`, `set`, ...).
- English texts contain $\approx$ **60% ambiguous tokens**.

Part-of-speech-Tagging (POS-Tagging)

= given a text, determine each token's POS.

- Core step for **many applications**: recognizing entities, parsing, key word detection ...
- 92% accuracy by simple baseline (choosing most frequent POS per word), 97% accuracy by **machine learning** *(later)*.

POS-Tagging in Python

```python
import nltk

text = "We watched the sunset over the castle on the hill."

tokens = nltk.word_tokenize(text)

result = nltk.pos_tag(tokens)

print(result)

> [('We', 'PRP'),              # OK (personal pronoun)
   ('watched', 'VBD'),         # OK (verb past tense)
   ('the', 'DT'),              # OK (determiner)
   ('sunset', 'NN'),           # OK (noun singular)
   ('over', 'IN'),             # OK (preposition)
   ('the', 'DT'),              # OK (determiner)
   ('castle', 'NN'),           # OK (noun singular)
   ('on', 'IN'),               # OK (preposition)
   ('the', 'DT'),              # OK (determiner)
   ('hill', 'NN'),             # OK (noun singular)
   ('.', '.')]
```

Outline

Statistical Word Similarity

Q: Which **insurance** company is based in **Wiesbahden**?
A: The R+V, a big **insurer**, resides in **Wiesbaden**.

- For QA (and many other applications), it is interesting to find **similar words**.
- Example: **query expansion**: (`laptop` → `notebook`).
- There are several forms of similarity *(semantic, statistical, syntactic)*.
- Sometimes, we can identify similar words by **syntactic** similarity (i.e., similarity of the word form).
- In other cases, this is not enough:

`insurance` → `insurer` ✓
`laptop` → `notebook` ↯

Statistical Word Similarity

Statistical Word Similarities

- **... learn** word similarity from a **text corpus**.
- Words are similar if their **contexts** are similar!

> *"You shall know a word by the company it keeps."*
>
> (J.R.Firth (1957))

Example: What other words is "tesgüino" similar to?

A bottle of **tesgüino** *is on the table.*
Tesgüino *makes you drunk.*
We make **tesgüino** *out of corn.*

Statistical Word Similarity

There are two kinds of statistical similarity

- **first-order** ("collocations"): w_2 is similar to w_1, if w_2 appears **in $\mathbf{w_1}$'s context** frequently *(e.g.* `wrote` → `book`*)*.
- **second-order**: w_2 is similar to w_1 if w_1 and w_2 tend to appear in **similar contexts** *(e.g.* `vodka` → `whisky`*)*.
- Context usually refers to a local window of $\pm(1-8)$ words.

Computing Statistical Word Similarities

- **Given**: vocabulary of n words + text corpus.
- **Goal**: compute an $n \times n$-matrix A (**term-term matrix**), such that A_{ij} is the similarity between w_i and w_j.
- Naive solution:

$$A_{ij} := \# \text{ occurrences of } w_j \text{ in the contexts of } w_i$$

- Why is this solution **bad**?

	wrote	book	...	carbon	dioxide	...	is	of
wrote	-	128		3	1		223	351
book		-		4	3		278	624
...			...					
carbon				-	46		98	75
dioxide					-		44	52
...						...		
is							-	9748
of								-

PPMI Word Similarity

Positive Pointwise Mutual Information (PPMI) ([3], Chapter 15)

Let w_i be a word, w_j be a "context word", and A the term-term matrix *(see above)*.
We calculate

- The probability of w_i appearing in a **random local word pair**:

$$P(w_i) := \sum_j A_{ij} / \sum_{i',j'} A_{i'j'}$$

- The probability of w_j appearing as a **context term**:

$$P(w_j) := \sum_i A_{ij} / \sum_{i',j'} A_{i'j'}$$

- The probability that **both words** appear in the same pair:

$$P(w_i, w_j) := A_{ij} / \sum_{i',j'} A_{i'j'}$$

PMI Word Similarity

- We know *(statistics)*: If both words are **independent**, then:

$$P(w_i, w_j) \approx P(w_i) \cdot P(w_j)$$

- We are interested in word pairs appearing **unusually frequently**, i.e.:

$$P(w_i, w_j) >> P(w_i) \cdot P(w_j)$$

- This leads to the definition of **Pointwise Mutual Information (PMI)**:

$$PMI(w_i, w_j) := log_2 \left(\frac{P(w_i, w_j)}{P(w_i) \cdot P(w_j)} \right)$$

PMI Word Similarity

- The **higher** PMI, the more **similar** two words.
- **Example**: The top-10 and bottom-10 pairs from Wikipedia.

word 1	word 2	count word 1	count word 2	count of co-occurrences	PMI
puerto	rico	1938	1311	1159	10.0349081703
hong	kong	2438	2694	2205	9.72831972408
los	angeles	3501	2808	2791	9.56067615065
carbon	dioxide	4265	1353	1032	9.09852946116
prize	laureate	5131	1676	1210	8.85870710982
san	francisco	5237	2477	1779	8.83305176711
nobel	prize	4098	5131	2498	8.68948811416
ice	hockey	5607	3002	1933	8.6555759741
star	trek	8264	1594	1489	8.63974676575
car	driver	5578	2749	1384	8.41470768304
it	the	283891	3293296	3347	-1.72037278119
are	of	234458	1761436	1019	-2.09254205335
this	the	199882	3293296	1211	-2.38612756961
is	of	565679	1761436	1562	-2.54614706831
and	of	1375396	1761436	2949	-2.79911817902
a	and	984442	1375396	1457	-2.92239510038
in	and	1187652	1375396	1537	-3.05660070757
to	and	1025659	1375396	1286	-3.08825363041
to	in	1025659	1187652	1066	-3.12911348956
of	and	1761436	1375396	1190	-3.70663100173

From PMI to PPMI

Since we are not interested in negative values, we clip them to zero, and obtain the **Positive Pointwise Mutual Information (PPMI)**:

$$PPMI(w_i, w_j) := max\left(log_2\left(\frac{P(w_i, w_j)}{P(w_i) \cdot P(w_j)}\right), 0\right)$$

	wrote	book	...	carbon	dioxide	...	is	of
wrote	-	**2.9**		0.5	0		0	0.1
book		-		0.2	0.2		0	0.1
...			...					
carbon				-	**7.2**		0.8	0
dioxide					-		0.1	0
...						...		
is							-	0
of								-

PPMI Word Similarity

Remarks

- **Challenge**: very frequent words that co-occur spuriously sometimes get **extreme PPMI values**. Therefore, we **smooth** the term-term matrix A before computing PPMI by adding a constant $(1 - 2)$.
- PPMI ist ein measure for **first-order**-similarity, but we can also compute second-order similarity from A [2].

Outline

1. Fundamentals

2. Preprocessing and Heuristics

3. Statistical Word Similarity

4. Syntactical Word Similarity

Syntactical Word Similarity: Edit Distances

Now: Syntactical Word Similarity

- … measures how similar two strings are in terms of their characters.
- useful for typos or variations of the same word *(above)*.
- Common approach: **string edit distances**.

Edit Distances

- Given two words/strings $\mathbf{s}_1, \mathbf{s}_2$ of length n_1, n_2, an **edit distance** corresponds to the **number of operations** required to transform $\mathbf{s}_1$ into $\mathbf{s}_2$.
- Different distances allow **different operations**:
  - **Levenshtein distance**:
    inserting/deleting/replacing 1 character.
  - **Damerau-Levenshtein distance**:
    also, transpose two adjacent characters.
  - ...

Levenshtein Distance: Examples

- $\mathrm{D}(\texttt{'typos'}, \texttt{'typo}\underline{\texttt{h}}\texttt{s'}) = 1$
- $\mathrm{D}(\texttt{'W}\underline{\texttt{ei}}\texttt{sba}\underline{\texttt{h}}\texttt{dn'}, \texttt{'Wiesbad}\underline{\texttt{e}}\texttt{n'}) = 4$

Computing Levenshtein: Dynamic Programming

insert **B** | **optimal path** | replace **G** with **B**

		B	**I**	**B**	**E**	**L**
	0	**1**	2	3	4	5
I	1	1	**1**	2	3	4
G	2	2	2	**2**	3	4
E	3	3	3	2	**2**	3
L	4	4	4	3	3	**2**

final distance

- Create a $(n_1 \times n_2)$ **cost matrix** D.
- Initialize D's first row (+column) with $0, 1, \ldots, n_1$ $(0, 1, \ldots, n_2)$.
- **traverse** D **row-wise**, and every row from **left to right**. Compute each cell (i, j):

$$D_{i,j} \leftarrow min\left(\underbrace{1 + D_{i-1,j}}_{delete},\ \underbrace{1 + D_{i,j-1}}_{insert},\ \underbrace{\mathbf{1}_{s_1(i) \neq s_2(j)} + D_{i-1,j-1}}_{replace} \right)$$

- Finally, D_{n_1,n_2} contains the Levenshtein distance.

Levenshtein Distance: Speed

- **Complexity**? → $O(n_1 \cdot n_2)$ ☹
- **Problem**: In practice, we want to compare a query q string/word with 100,000s of other words!
- **Example**: query=`wiesbahden`, find the most similar vocabulary word to correct the typo.

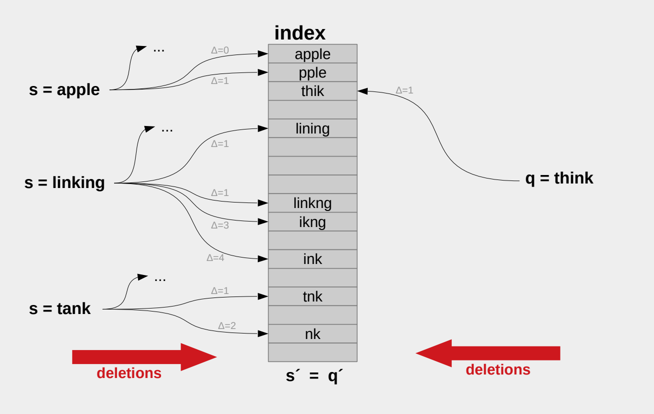
Approach

Build **index structures** for fast term similarity matching:

1. Symspell ("reduction indexing")
2. Min Hashing

Symspell

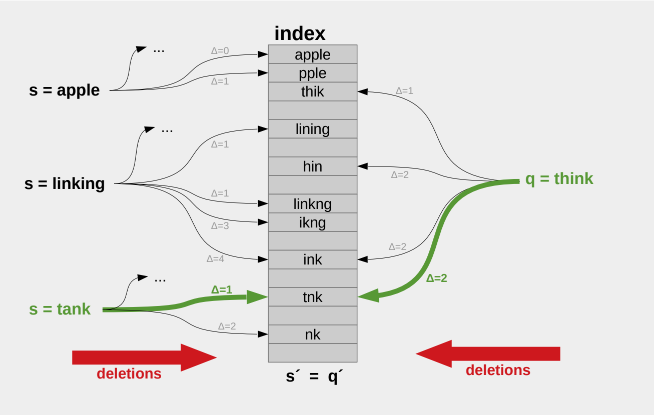
- a method for fast string indexing: Symspell by Wolf Garbe[4].
- Step 1: For each string s in the vocabulary: delete $\Delta = 1, 2, \ldots, K$ characters and store the resulting versions s' in a hash map.



[4] https://github.com/wolfgarbe/SymSpell (C#), https://github.com/mammothb/symspellpy (Python)

Symspell

- Step 2: Given a query q, perform the same delete operations, and lookup all reduced queries q' in the hash map.
- If $q' = s'$, we know that $D(q,s) \leq \Delta(s,s') + \Delta(q,q')$.
- In the example: $D(think, tank) \leq \Delta(think, tnk) + \Delta(tnk, think) = 3$ *(in fact, $D(think, tank) = 2$).*



Symspell

Remarks

- Huge efficiency gain, partiularly because replaces and inserts are avoided *(think languages with huge alphabets like Chinese)*.
- For each candidate s' we find, we can compute the exact distance $D(q, s)$.
- The reduced versions q' are iterated by increasing the number of deletes. Thus, we can stop once we cannot find a better version than the best match so far.

References I

[1] Marti A. Hearst.
Automatic acquisition of hyponyms from large text corpora.
In Proceedings of the 14th Conference on Computational Linguistics - Volume 2, COLING '92, pages 539–545, Stroudsburg, PA, USA, 1992. Association for Computational Linguistics.

[2] A. Islam and D. Inkpen.
Second order co-occurrence PMI for determining the semantic similarity of words.
In Proc. LREC 2006, pages 1033–1038, 2006.

[3] Daniel Jurafsky and James H. Martin.
Speech and Language Processing: An Introduction to Natural Language Processing (3rd Edition Draft Chapters).
2017.