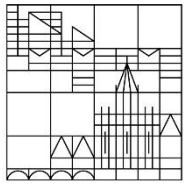
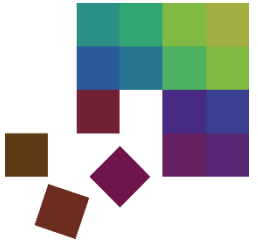


University of Konstanz
Data Analysis and Visualization Group



Text Processing

Raphael Buchmüller

Computational Linguistics Fall School 2024

Whats the Schedule?

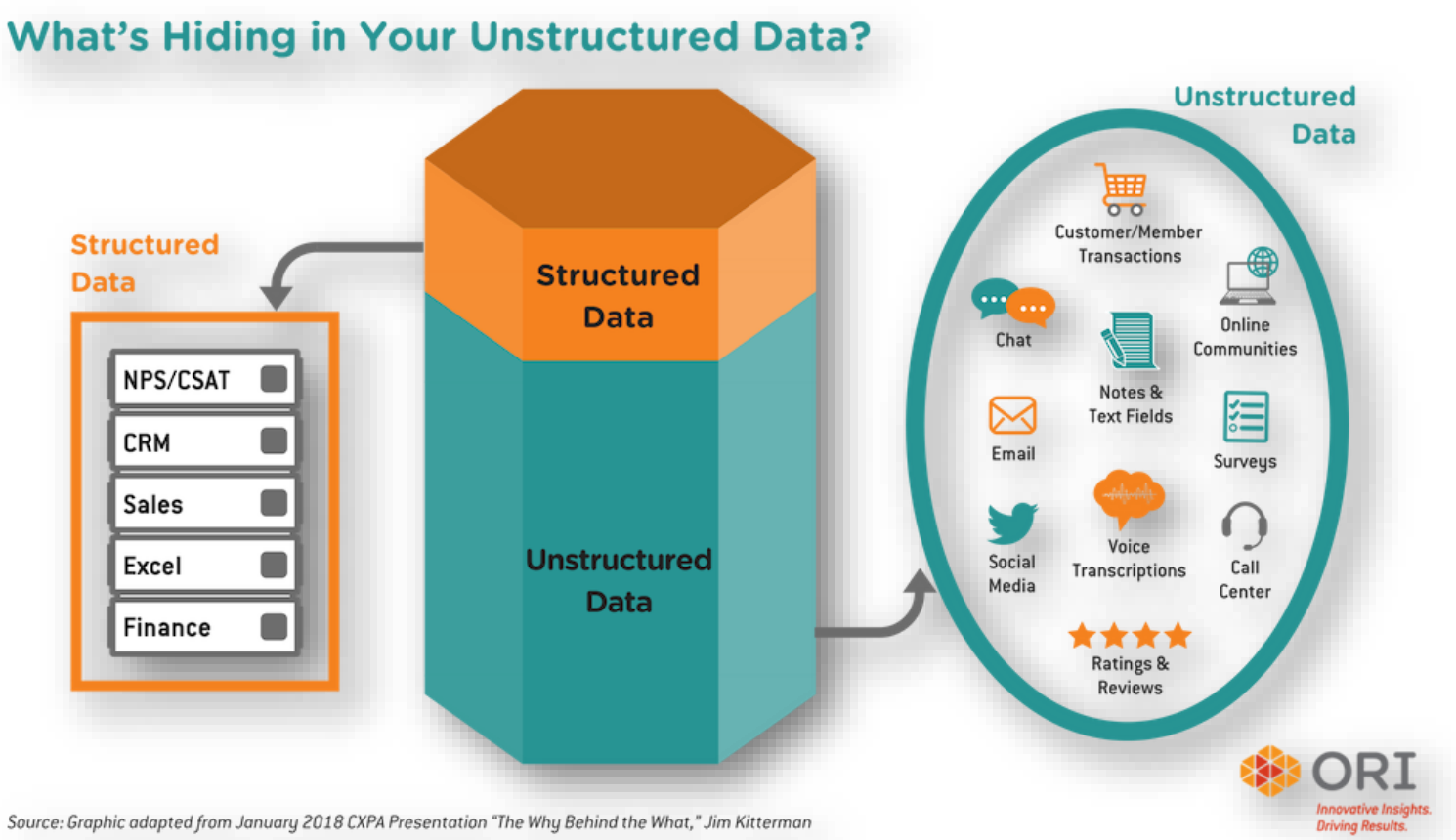
	Theory (1:30-3pm)	Practice (3:30-5pm)
Monday	Domain and Design	Design your Approach
Tuesday	**Text Processing**	Process your Text
Wednesday	Visualization Foundations	Implementation 1
Thursday	Text Visualization	Implementation 2
Friday	Projects, Experiences and Discussion	Present your Approach

Text Processing

- Introduction: Why is NLP difficult?
- Elements of Language
- Basic NLP
- Named Entity Reckognition
- Similarity and Search
- Word Embeddings

Need for Computational Methods

About 80% of the information in organizations is made up by textual data.



Unstructured Data

- Typically refers to free text
- Classic model for searching text documents:
  - Information Retrieval
- Allows
  - Keyword queries including operators
  - More sophisticated “concept” queries e.g.,
    - find all web pages dealing with *drug abuse*

Semi-Structured Data

- In fact almost no data is “unstructured”
- E.g., this slide has distinctly identified zones such as the *Title* and *Bullets*
- Facilitates “semi-structured” search such as
  - *Title* contains data AND *Bullets* contain search

Applications of Computational Methods

- Application areas include

Biomedical research

- Information retrieval in biomedical literature databases (e.g. PubMed)

Marketing

- Analysis for customer relationship management
- Opinion Mining / Sentiment Analysis / Product Analysis
- Customer Feedback Analysis

Security

- E.g. uncovering criminal or terrorist activities

Applications of Computational Methods

- Application areas include

Automation of Document Processing
- Automatic Document Sorting and Enhancement
- Automatic Response

Literature Analysis
- Authorship Attribution
- Readability Evaluation

Linguistic and Political Sciences Research
- Discourse Analysis
- Analysis of Language Development
- Argumentation Mining

...

Why is NLP difficult?

Why is NLP difficult?

```
SUBJECT PUNCHED QUICKLY OXIDIZED TCEJBUS DEHCNUP YLKCIUQ DEZIDIXO
 CERTAIN QUICKLY PUNCHED METHODS NIATREC YLKCIUQ DEHCNUP SDOHTEM
 SCIENCE ENGLISH RECORDS COLUMNS ECNEICS HSILGNE SDROCER SNMULOC
 GOVERNS PRECISE EXAMPLE MERCURY SNREVOG ESICERP ELPMAXE YRUCREM
 CERTAIN QUICKLY PUNCHED METHODS NIATREC YLKCIUQ DEHCNUP SDOHTEM
 GOVERNS PRECISE EXAMPLE MERCURY SNREVOG ESICERP ELPMAXE YRUCREM
 SCIENCE ENGLISH RECORDS COLUMNS ECNEICS HSILGNE SDROCER SNMULOC
SUBJECT PUNCHED QUICKLY OXIDIZED TCEJBUS DEHCNUP YLKCIUQ DEZIDIXO
 CERTAIN QUICKLY PUNCHED METHODS NIATREC YLKCIUQ DEHCNUP SDOHTEM
 SCIENCE ENGLISH RECORDS COLUMNS ECNEICS HSILGNE SDROCER SNMULOC
```

Why is NLP difficult?

Why is NLP difficult?

The dog.

Why is NLP difficult?

The dog.

The dog cavorts.

The dog cavorted.

Why is NLP difficult?

Why is NLP difficult?

The man.

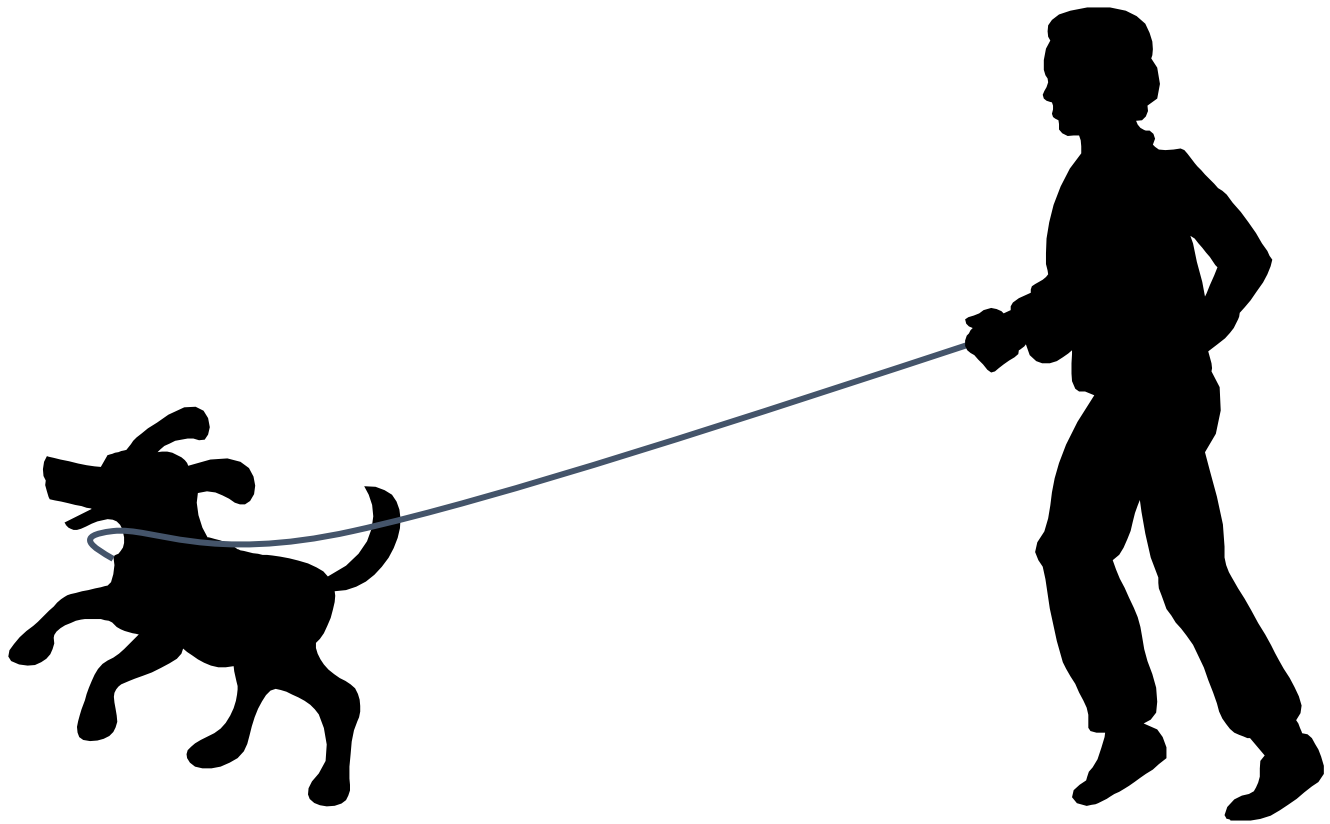
Why is NLP difficult?

The man.

The man walks.

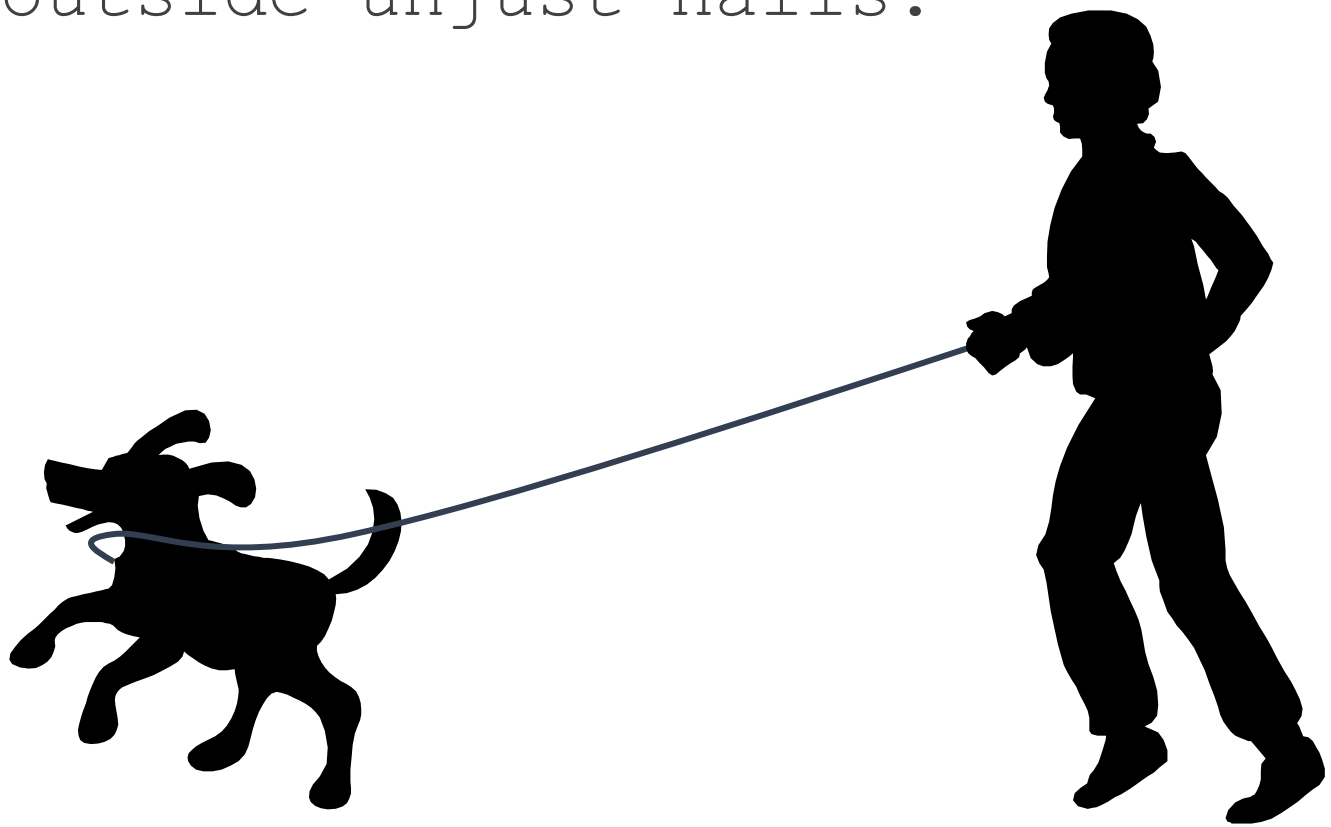
Why is NLP difficult?

The man walks the cavorting dog.



Why is NLP difficult?

As the man walks the cavorting dog, thoughts arrive unbidden of the previous spring, so unlike this one, in which walking was marching and dogs were baleful sentinels outside unjust halls.



Why is NLP difficult?

- Computers have no brains
  - There is evidence that much of language understanding is built-in to the human brain
- Computers do not socialize
  - Much of language is about communicating with people

Why is NLP difficult?

- Key problems:
  - Representation of *meaning*
  - Language presupposes knowledge about the world
  - Language only reflects the surface of meaning
  - Language presupposes communication between people

Why is NLP difficult?

- Example:

„How much is the doggy in the window?“

Why is NLP difficult?

- Example:
  - „How much“: is not the size or the weight but the costs

„How much is the doggy in the window?“

Why is NLP difficult?

- Example:
  - „the doggy“: is childlike, probably cannot purchase on their own

„How much is the doggy in the window?“

Why is NLP difficult?

- Example:
  - „the window“: behind a store not inside a window.

„How much is the doggy in the window?“

How can a machine understand these differences?

Get the cat with the gloves.

Text Ambiguities – Metaphorical Language

Time flies like an arrow.

Think about the many different meanings…

Text Ambiguities

Every American has a president.

Text Ambiguities

Every American has a mother.

Text Ambiguities

We gave the monkeys the bananas because they were hungry.

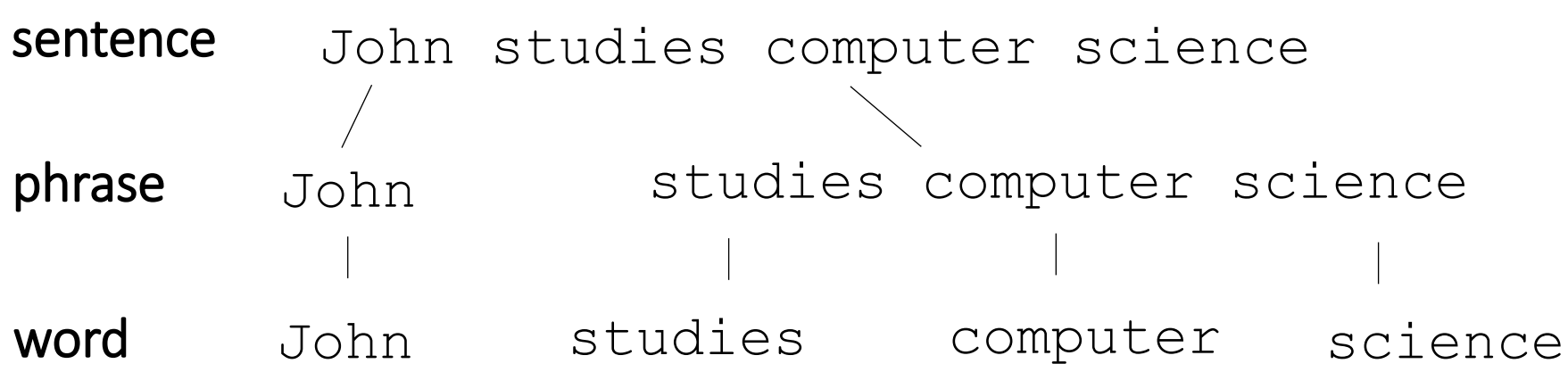
Text Ambiguities

We gave the monkeys the bananas because they were over-ripe.

Elements of Language

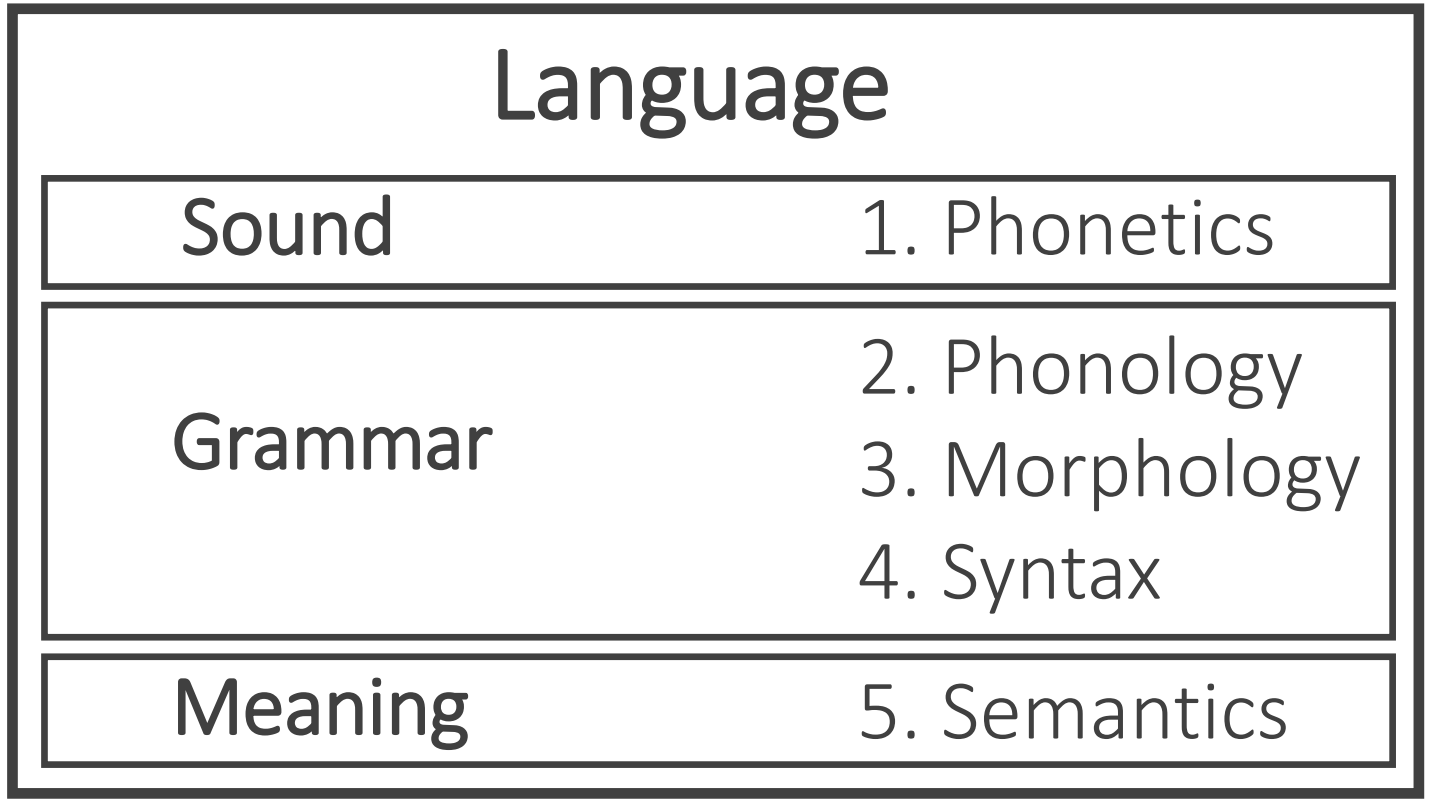
Elements of Language

- NLP hierarchy:

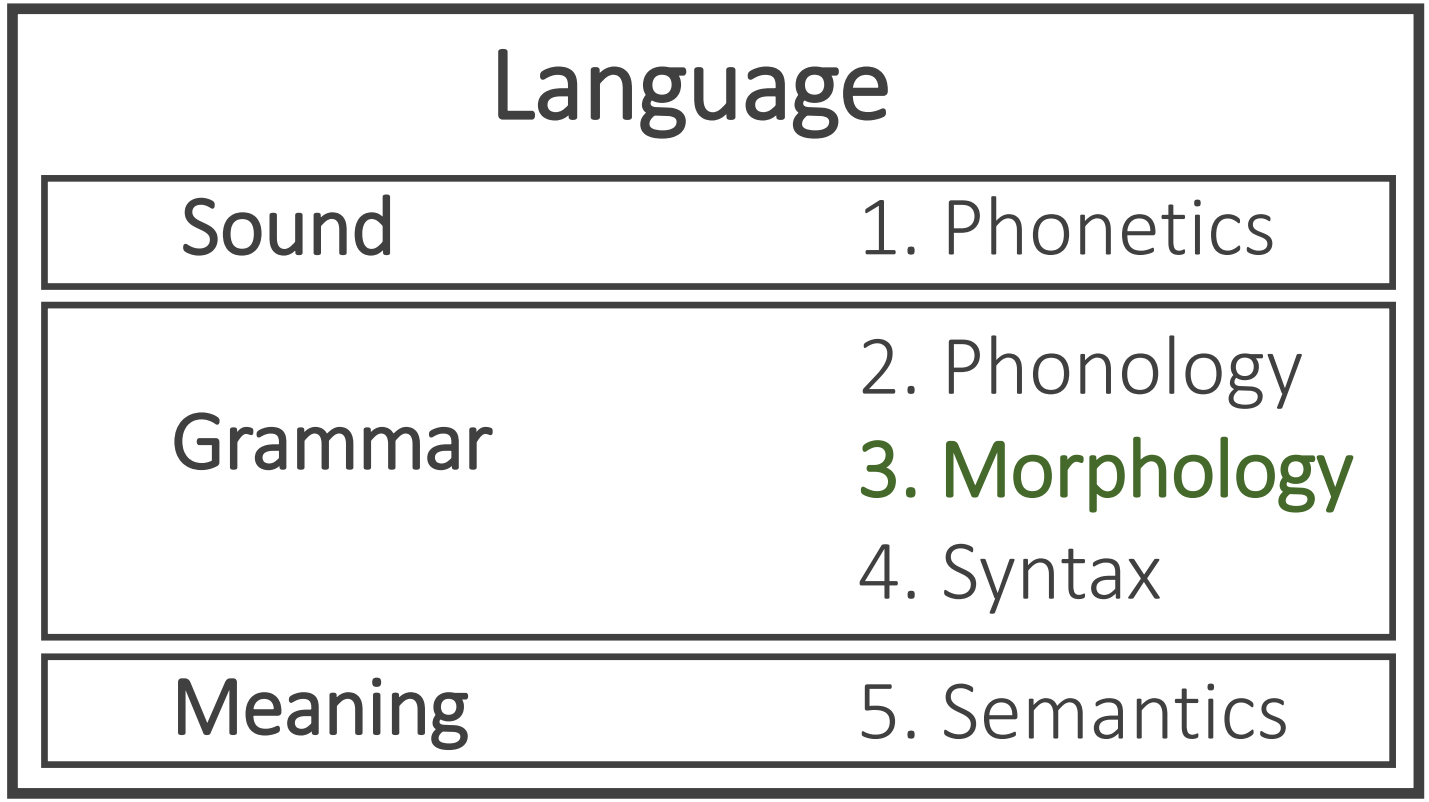


- Words can be analyzed further:
  - Phonemes: `studies` → `s-t-u-d-i-e-s`
  - Morphemes: `studies` → `studi-es`
  - Syllables: `studies` → `stud-ies`
  - Stem: `studies` → `studi`
  - Lemma: `studies` → `study`

Elements of Language



Elements of Language



Jabberwocky Analysis

- This is nonsense … or is it?
- This is not English …
  but it's much more like English
  than it is like French or German
  or Chinese or …

JABBERWOCKY

Lewis Carroll

(from *Through the Looking-Glass and What Alice Found There*, 1872)

`Twas brillig, and the slithy toves
Did gyre and gimble in the wabe:
All mimsy were the borogoves,
And the mome raths outgrabe.

"Beware the Jabberwock, my son!
The jaws that bite, the claws that catch!
Beware the Jubjub bird, and shun
The frumious Bandersnatch!"

He took his vorpal sword in hand:
Long time the manxome foe he sought --
So rested he by the Tumtum tree,
And stood awhile in thought.

And, as in uffish thought he stood,
The Jabberwock, with eyes of flame,
Came whiffling through the tulgey wood,
And burbled as it came!

Jabberwocky Analysis

- Why do we pretty much understand the words?
- We recognize combinations of morphemes.

  - `Chortled` - *Laugh in a breathy, gleeful way;* (Definition from Oxford American Dictionary)
    A combination of "chuckle" and "snort."

  - `Galumphing` - *Moving in a clumsy, ponderous, or noisy manner.* (Definition from Oxford American Dictionary)
    Perhaps a blend of "gallop" and "triumph."

Morphology

Morphology:

- The study of the way words are built up from smaller meaning units.

Morphemes:

- The smallest meaningful unit in the grammar of a language.

A useful resource:

- Glossary of linguistic terms by Eugene Loos
- http://www.sil.org/

Morpheme Examples

`unladylike`

- 3 morphemes, 4 syllables

  `un-` ‘not’

  `lady` ‘(well behaved) female adult human’

  `-like` ‘having the characteristics of’

- Can’t break any of these further down without distorting the meaning of the units

- **`technique`**
  - 1 morpheme, 2 syllables
- **`dogs`**
  - 2 morphemes, 1 syllable

    -s, a plural marker on nouns

Learning a Language by Morpheme Analysis

Example sentences from Swahili, East Africa

1.	ninasema.	I speak.
2.	unasema.	You *(sg)* speak.
3.	anasema.	He speaks.
4.	wanasema.	They speak.
5.	ninaona.	I see.
6.	niliona.	I saw.
7.	ninawaona.	I see you *(pl)*.
8.	nilikuona.	I saw you *(sg)*.
9.	ananiona.	He sees me.
10.	utaniona.	You *(sg)* will see me.

A) Try to identify the morphemes of the language + their meaning.

B) What does „nilisema“ mean?

sg: singular, pl: plural

Learning a Language by Morpheme Analysis

Example sentences from Swahili, East Africa

1. ninasema.	I speak.
2. unasema.	You *(sg)* speak.
3. anasema.	He speaks.
4. wanasema.	They speak.
5. ninaona.	I see.
6. niliona.	I saw.
7. ninawaona.	I see you *(pl)*.
8. nilikuona.	I saw you *(sg)*.
9. ananiona.	He sees me.
10. utaniona.	You *(sg)* will see me.

A) Try to identify the morphemes of the language + their meaning.

B) What does „nilisema“ mean?

C) Can you translate the following sentences?

1. You *(sg)* see me.
2. He saw them.
3. I will see you.

sg: singular, pl: plural

Tokenization

Tokens vs. Types

The term *word* can be used in two different ways:

1. To refer to an individual occurrence of a word
2. To refer to an abstract vocabulary item
   - For example, the sentence "*my dog likes his dog*" contains five occurrences of words, but four vocabulary items.

To avoid confusion we use a more precise terminology:

1. **Word Token***:* an occurrence of a word
2. **Word Type***:* a vocabulary item

Tokenization

- **Tokenization**: segmentation of an input stream into an ordered sequence of tokens

- **Tokenizer**: a system which splits texts into word tokens

A very simple example:

- Input text:
  `John likes Mary and Mary likes John.`
- Tokens:
  `{"John", "likes", "Mary", "and", "Mary", "likes", "John", "."}`

Tokenization

Simple approach

- Split sentences at punctuation marks
- Split tokens at whitespace characters

Example

- Mr. Sherwood said, reaction to Sea Containers‘ proposal has been „very positive.“ In New York Stock Exchange composite trading yesterday, Sea Containers closed at $62.625, up 62.5 cents.

- Split at whitespace characters?

  said, positive.” $62.625, cents.

Tokenization Ambiguities

Period

- In most of the cases: Final sentence punctuation symbol
- Part of an abbreviation, e.g. `F.D.P.`
- Numbers, ordinal numbers, e.g.: `21.`, numbers with fractions, e.g. `1.543`
- References to resource locators, e.g.: `www.apple.com`
- …

Whitespace character

- Part of numbers, e.g. `1 543`
- No segmentation character in multi-word expressions, e.g. `New York`

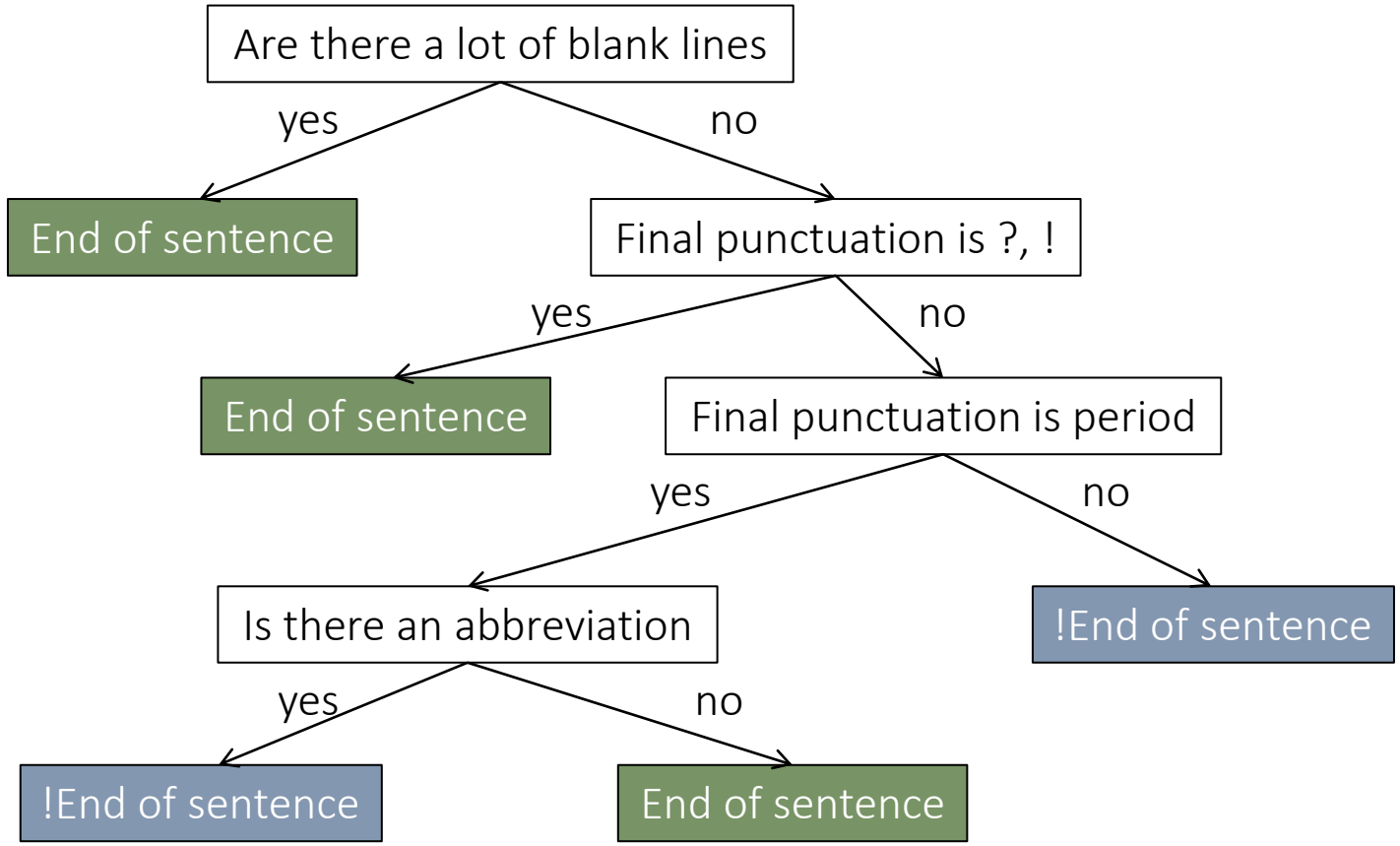
Similar issues with **commas, dashes, single quotes**

Sentence Detector

- How to start?
- !, ? Are relatively unambiguous
- Period „.“ is quite ambiguous
  - Sentence boundary
  - Abbrevations
  - Numbers
  - …
- Build a binary classifier!
  - (End of sentence or not)

Sentence Detector

- You could build a decision tree



Tokenization: Special Cases

- Finland‘s capital -> Finland Finlands Finland ‘s
- What‘re, I‘m, isn‘t -> What are, I am, is not
- Hewlett-Packard -> Hewlett Packard
- State-of-the-art -> state of the art
- Lowercase -> lower-case, lowercase, lower case
- San Francisco -> one token or two
- PhD. -> ??

From the Stanford Coursera course by Dan Jurafsky and Christopher Manning

Tokenization in German

German:

- „Lebensversicherungsgesellschaftsangestellter“
- „life insurance company employee“
- Therefore, we need a compound splitter for the german language in case of information retrieval

Tokenization in Other Languages

Chinese:

乒乓球拍卖完了。

- No spaces
- Two possible segmentations, both of them are syntactically and semantically correct
- Disambiguation can only be done with contextual information

乒乓 /球拍 /卖完了

ping-pong racket sold out

(The ping-pong rackets have been sold out.)

乒乓球 /拍卖 /完了。

ping-pong ball auction finish

(The auction of the ping-pong ball has been finished.)

Libraries

- In practice, use one of the available NLP libraries
- But, different libraries may produce different results!

	White spaces	Apache Open NLP 1.8.3 (Model)	Apache Open NLP 1.8.3 (Rules)	Stanford NLTK 3.8	Optimal
1		“	“	“	“
2	“I	I	I	I	I
3	said,	said	said	said	said
4		,	,	,	,
5			‘	‘	‘
6	‘what‘re	‘what	what	what	what
7			‘		
8		‘re	re	‘re	are
9	you?	you	you	you	you
10		?	?	?	?
11	Crazy?‘“	Crazy	Crazy	Crazy	Crazy
12		?	?	?	?
13		‘	‘	‘	‘
14		“	“	“	“
15	said	said	said	said	said
16	Sandowsky.	Sandowsky	Sandowsky	Sandowsky	Sandowsky
17		.	.	.	.
18		“	“	“	“
19	“I	I	I	I	I
20	can‘t	ca	can	ca	can
21			‘		
22		n‘t	t	n‘t	not
23	afford	afford	afford	afford	afford
24	to	to	to	to	to
25	do	do	do	do	do
26	that.“	that	that	that	that
27		.	.	.	.
28		“	“	“	“

Stemming

Stemming

- Reduction of a word to its stem
- Input text:

```
John Vinken , 61 years old , will join the board as a
nonexecutive director Nov. 29 .
```

- Stems:

```
John Vinken , 61 year old , will join the board as a
nonexecut director Nov. 29 .
```

Outline

- Why is NLP difficult?
- Elements of Language
- Tokenization
- **Stemming**
- Part-of-Speech Tagging
- Parsing

Stemming

- Tokenization ideally deliver a list of words
- To make these words comparable, we want to reduce them to their root morpheme (remember: computerization -> comput-er-ization)
- We can use stemming to reduce a word to its stem (comput) or lemmatization to reduce a word to its lemma (compute)

→ Now we see that {computerization, computer, computing, computation} all belong to the same root

Porter Stemmer

- A simple approach: just hack off the end of the word!
- Frequently used, especially for Information Retrieval, but results are pretty ugly (from a linguistic perspective)!

- rules for suffix stripping are applied:
  `sses → s, ies → i, s → _`

Porter Stemmer

- **Only rules for suffixes** not for prefixes.
- Rule-based approach
  - Example 1:
    - Input = `computational`
    - Output = `comput`
  - Example 2:
    - Input = `computer`
    - Output = `comput`

Online Tools

http://text-processing.com/demo/

http://textanalysisonline.com/nltk-wordnet-word-lemmatizer

http://textanalysisonline.com/nltk-wordnet-lemmatizer

POS Tagging

Part-of-Speech Tagging

- Marking up a word in a text as corresponding to a particular part of speech
- Input text:

```
John is happy
```

- POS tags:

```
John <PROPN> is <VB> happy <JJ>
```

Significance of Parts of Speech

- A word's POS tells us a lot about the word and its neighbors:
  - Limits the range of meanings (*deal*), pronunciation (*object vs object*) or both (*wind*)

Significance of Parts of Speech

- A word's POS tells us a lot about the word and its neighbors:
  - Limits the range of meanings (*deal*), pronunciation (*object vs object*) or both (*wind*)
  - Helps in stemming

Significance of Parts of Speech

- A word's POS tells us a lot about the word and its neighbors:
  - Limits the range of meanings (*deal*), pronunciation (*object vs object*) or both (*wind*)
  - Helps in stemming
  - Limits the range of following words

Significance of Parts of Speech

- A word's POS tells us a lot about the word and its neighbors:
  - Limits the range of meanings (*deal*), pronunciation (*object vs object*) or both (*wind*)
  - Helps in stemming
  - Limits the range of following words
  - Can help select nouns from a document for summarization

Significance of Parts of Speech

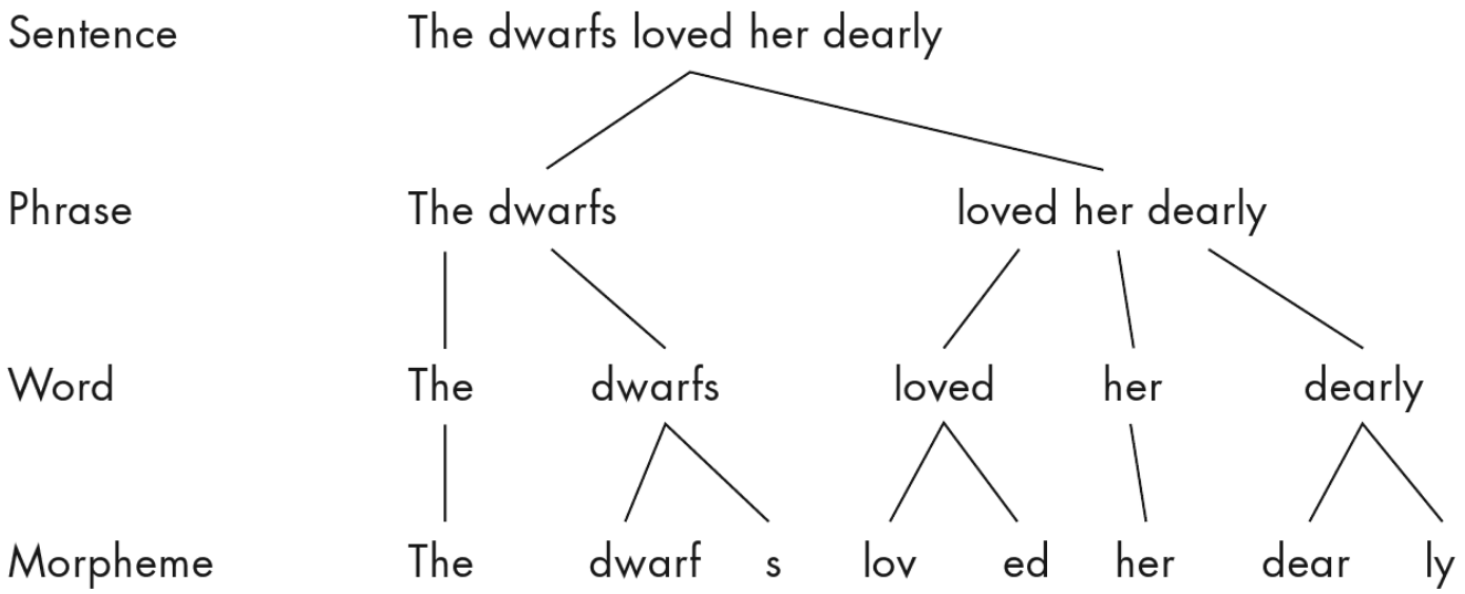
- A word's POS tells us a lot about the word and its neighbors:
  - Limits the range of meanings (*deal*), pronunciation (*object vs object*) or both (*wind*)
  - Helps in stemming
  - Limits the range of following words
  - Can help select nouns from a document for summarization
  - Basis for partial parsing (chunked parsing)
  - Parsers can build trees directly on the POS tags instead of maintaining a lexicon

List of POS tags
https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

Parsing

What is Parsing?

- The process of **determining the grammatical structure** with respect to a given grammar.



Applications of Parsing

- Grammar checking
  - `I want to return this shoes`
- Question answering
  - `How many people in sales make 40k or more per year?`
- Machine translation
  - E.g., word order: SVO vs. SOV
- …

Syntax

- Language is not just a “bag of words”
- There are grammatical rules
  - Do not apply to individual words
  - …but to categories or group of words

- Example:
  - A sentence includes a subject and a predicate. The subject is a noun phrase and the predicate is a verb phrase:
  - Noun phrase: `The cat`, `Samantha`, `She`
  - Verb phrase: `arrived`, `went away`, `had dinner`

- When people learn a new word, they learn its syntactical usage

Named Entity Annotation

Named Entity Annotation

Person

Location

Organization

Number

Time / Date

Common Types of Named Entities

The core set (by definition):

- Persons (e.g., `Elon Musk`)
- Locations (e.g., `Konstanz`)
- Organizations (e.g., `EU`)

Further useful named entity types:

- Dates (e.g., `Friday, 13.05.`)
- Times (e.g., `13:37`)
- Numeric expressions (e.g., `$43bn`)

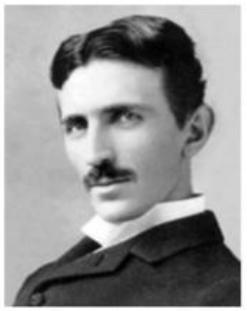
Domain-specific entity types:

- Chemicals (e.g., C_2H_5OH)
- Genes (e.g., `TP53`)
- Stock symbols (e.g., `AAPL`)
- Laws (e.g., `StGB`)
- URLs (e.g., `www.uni.kn`)
- etc...

Named Entity Ambiguity

Example:

The Tesla chief executive revealed on Thursday that he had launched a hostile takeover of Twitter.



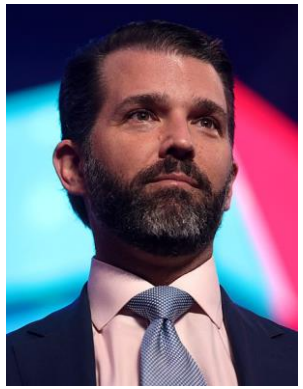
$$T = \frac{Vs}{m^2}$$

Company? Person? Unit?

Named Entity Normalization

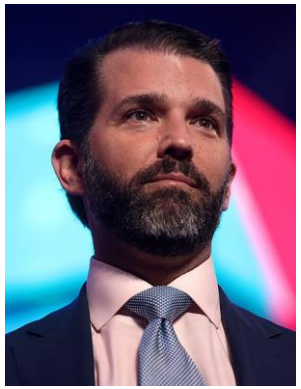
Donald John Trump Jr. is an American political activist, businessman, conspiracy theorist, author, and former television presenter. He is the eldest child of the 45th president of the United States, Donald J. Trump.

NE disambiguation (NED): The task of deciding whether two entity mentions refer to the same entity.



≠

NE linking (NEL): The task of linking an entity mention to a unique identifier.



Word Similarity

Word Similarity: Experimental Results

People were asked to rate the similarity of two words:

- Scale: 0 (not similar) - 10 (similar)
- Result: no consistent agreement between human annotators

⇨ When are two words similar?

word 1	word 2	sim
word 1	word 2	sim
tiger	cat	7.35
tiger	tiger	10.00
book	paper	7.46
computer	keyboard	7.62
computer	internet	7.58
plane	car	5.77
train	car	6.31
telephone	communication	7.50
television	radio	6.77
media	radio	7.42
drug	abuse	6.85
bread	butter	6.19
cucumber	potato	5.92

Finkelstein et al. *Placing search in context: The concept revisited*. International Conference on World Wide Web (2001)

Types of Similarity in Language

Surface form similarity:

- Phonological similarity (e.g., `brake | break`)
- Morphological similarity (e.g., `respect | respectful`)
- Spelling similarity (e.g. `theater | theatre`)

Semantic similarity:

- Synonymy (e.g. `verbose | wordy`)
- Hyponymy (e.g., `color | red`)

Content similarity:

- Sentence similarity (e.g., paraphrases)
- Document similarity (e.g., two news stories on the same event)

Surface Form Similarity

Spelling Similarity: Spell Checking

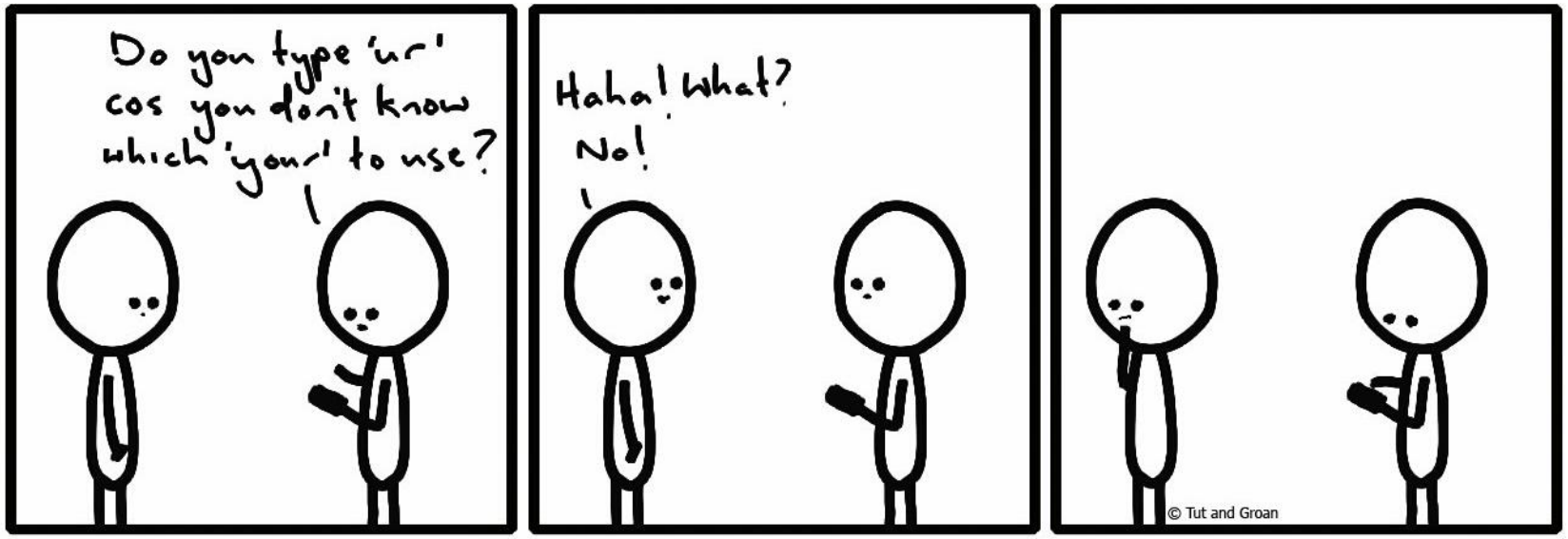
Principal uses

- Spell checking suggestions (e.g. `spell chekcing`)
- Correcting documents in a collection to improve corpus quality
- Retrieving matching documents when the query contains a spelling error

Spelling Similarity: Spell Checking

The are two flavors of spell checking:

- Isolated words
  - Check each word on its own for misspelling
  - Will not catch typos resulting in correctly spelled words (e.g., `from` and `form`)
- Context-sensitive
  - Look at the context provided by surrounding tokens
  - Example: `I flew form Heathrow to Zürich`



Levenshtein Distance

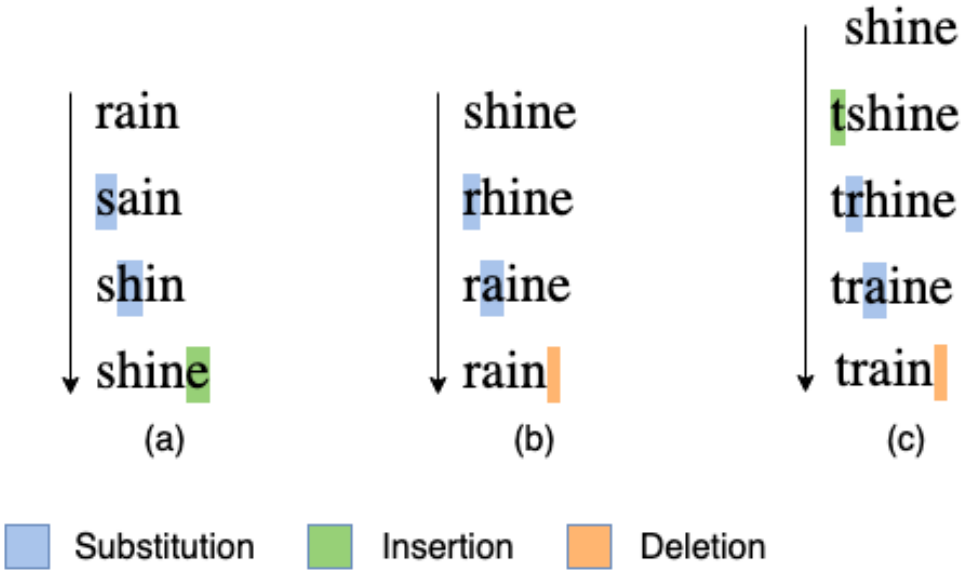
Idea behind the Levenshtein Distance (also called edit distance): Given two strings `S1` and `S2`, count the minimum number of basic operations to convert one string to the other.

Basic operations are typically character-level:

- Insert
- Delete
- Replace (i.e., substitute)

Example:

- The edit distance between `rain` and `shine` is 3
- We need to replace two characters and insert one character



Levenshtein Distance: Wagner-Fischer Algorithm (1)

Step 1: Setup and parameters
Set `n` to be the length of string `s`
Set `m` to be the length of string `t`
If `n=0`, return `m` and exit.
In `m=0`, return `n` and exit.
Construct a matrix containing `0…m` rows and `0…n` columns.

Step 2: Initialization
Initialize the zeroth row to `0…n`
Initialize the zeroth column to `0…m`

Example:

		c	a	t
	0	1	2	3
h	1			
a	2			
t	3			

`s = cat, n=3`

`t = hat, m=3`

Levenshtein Distance: Wagner-Fischer Algorithm (2)

Example:

		c	a	t
	0	1	2	3
h	1	1	2	3
a	2	2	1	2
t	3	3	2	1

Step 3: Iteration

```
For j from 1…n            // iterate over columns
   For i from 1…m         // iterate over rows

      if s[i] = t[j] then subCost := 0  // retain character
      if s[i] ≠ t[j] then subCost := 1  // replace character

      d[i, j] := minimum(d[i-1,   j] + 1,          // deletion
                         d[i,   j-1] + 1,          // insertion
                         d[i-1, j-1] + subCost)    // substitution
return d[m, n]
```

Wagner-Fischer Algorithm: Example

		S	T	R	E	N	G	T	H
	0	1	2	3	4	5	6	7	8
T	1								
R	2								
E	3								
N	4								
D	5								

Wagner-Fischer Algorithm: Example

		S	T	R	E	N	G	T	H
	0	1	2	3	4	5	6	7	8
T	1	1							
R	2								
E	3								
N	4								
D	5								

Parameters and choices:

[1,1] S ≠ T →

subCost = 1

delete: 1+1=2

insert: 1+1=2

substitute: 0+1=1

→ substitute is cheapest

Wagner-Fischer Algorithm: Example

		S	T	R	E	N	G	T	H
	0	1	2	3	4	5	6	7	8
T	1	1	1						
R	2								
E	3								
N	4								
D	5								

Parameters and choices:

[1,2] T = T →

subCost = 0

delete: 2+1=3

insert: 1+1=2

substitute: 1+0=1

→ substitute is cheapest

Wagner-Fischer Algorithm: Example

		S	T	R	E	N	G	T	H
	0	1	2	3	4	5	6	7	8
T	1	1	1	2					
R	2								
E	3								
N	4								
D	5								

Parameters and choices:

[1,3] R ≠ T →

subCost = 1

delete: 3+1=4

insert: 1+1=2

substitute: 2+1=3

→ insert is cheapest

Wagner-Fischer Algorithm: Example

		S	T	R	E	N	G	T	H
	0	1	2	3	4	5	6	7	8
T	1	1	1	2	3	4	5	5	6
R	2	2							
E	3								
N	4								
D	5								

Parameters and choices:

[2,1] S ≠ R →

subCost = 1

delete: 1+1=2

insert: 2+1=3

substitute: 1+1=2

→ delete and substitute are both equally cheap

Wagner-Fischer Algorithm: Example

		S	T	R	E	N	G	T	H
	0	1	2	3	4	5	6	7	8
T	1	1	1	2	3	4	5	5	6
R	2	2	2	1					
E	3								
N	4								
D	5								

Parameters and choices:

[2,3] R = R →

subCost = 0

delete: 2+1=3

insert: 2+1=3

substitute: 1+0=1

→ substitute is cheapest

Wagner-Fischer Algorithm: Example

		S	T	R	E	N	G	T	H
	0	1	2	3	4	5	6	7	8
T	1	1	1	2	3	4	5	5	6
R	2	2	2	1	2	3	4	5	6
E	3	3	3	2	1	2	3	4	5
N	4	4	4	3	2	1	2	3	4
D	5	5	5	4	3	2	2	3	4

Output:

The Levenshtein distance is 4

STrend: insert S at pos 0
Strend: retain T at pos 1
Strend: retain R at pos 2
Strend: retain E at pos 3
Strend: retain N at pos 4
Streng: replace D with G at pos 5
Strengt: insert T at pos 6
Strength: insert H at pos 7

Weighted Edit Distance

Core idea:

- Different costs for insert / delete operations
- Can use domain-specific knowledge

Damerau modification:

- Swaps of two adjacent characters also have a cost of 1
- Example:
  - `Lev(cats, cast) = 2`
  - `Dam(cats, cast) = 1`

Specialized Edit Distances

Consider these edit distances. Why (or where) could they make sense?

```
DistA(sit down, sit clown) = 1
```


THE PENNY SATIRIST.

No. 391. FOR THE WEEK ENDING SATURDAY, OCTOBER 12, 1844. Vol. 8.

EXTRAORDINARY FEAT; ASTLEY'S CLOWN OUTDONE.

```
DistB(qeather, weather) = 1
DistB(leather, weather) = 2
```

`DistA` models typical OCR errors that occur when processing old newspapers.

`DistB` models typing errors as a result of using a keyboard (= fat fingers)

Semantic Similarity

Semantic Similarity: Example

The S&P 500 climbed 6.9%, to 1,237, its best close since June 12, 2001.

The Nasdaq gained 12.2%, to 2,123 for its best showing since June 8, 2001.

The DJIA rose 68.46%, to 10,723, its highest level since March 15.

Semantic Relations

Synonymy:

- Different words with similar meaning (e.g., `big | large`)
- Synonyms differ in their frequency of use and the context

Antonymy:

- Words that are near opposites (e.g., `raise | lower`)

Hypernymy:

- Supertype of a word (e.g., `red` is a `color`)

Hyponymy

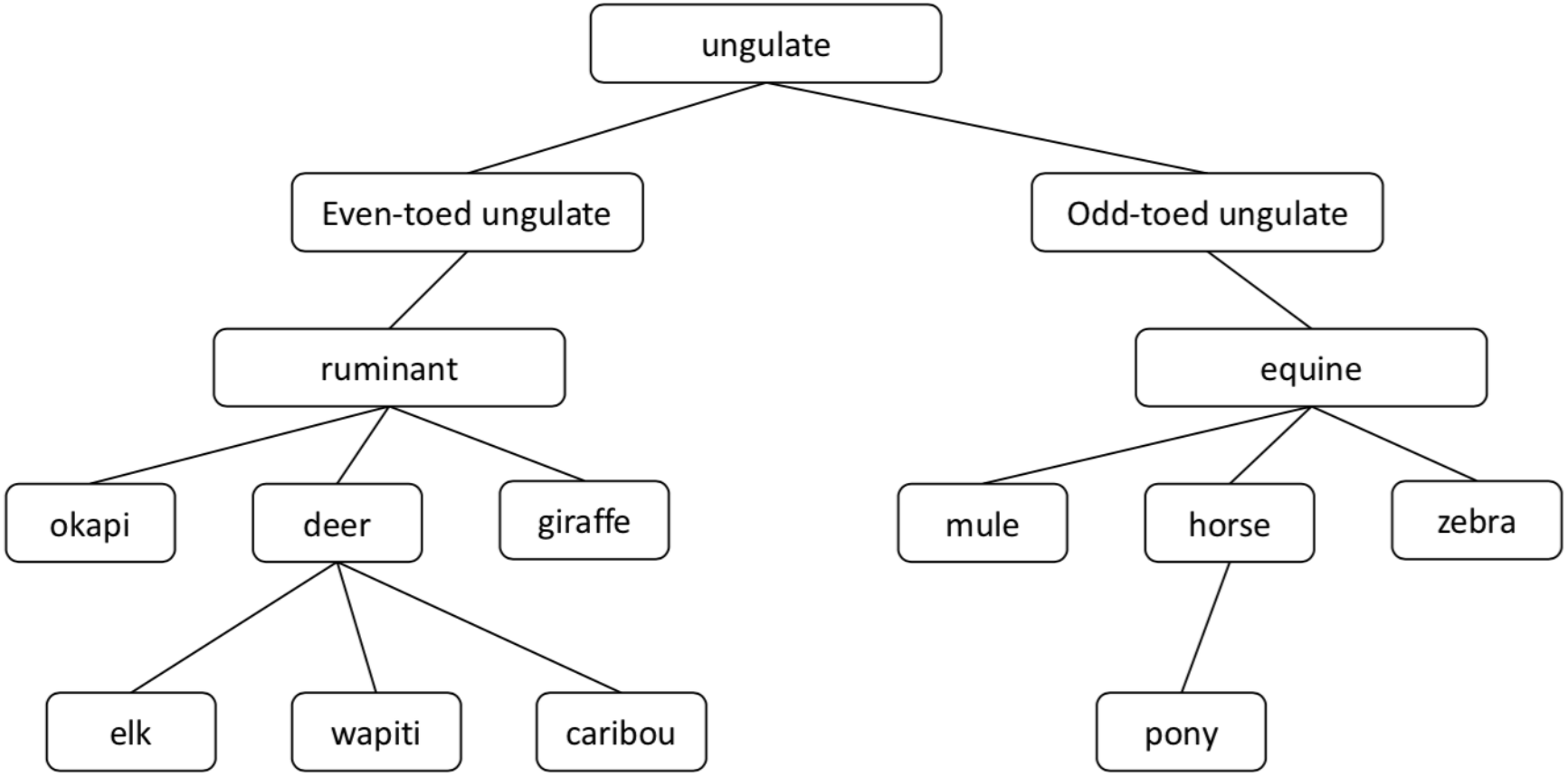
- Subtype of a word (inverse of hypernymy)

Meronymy:

- A word is part of a larger whole (e.g., a `flock of sheep` includes `sheep`)

WordNet

WordNet is a database of words and semantic relations between them. The main relation is hypernymy, so the overall structure is tree-like.



WordNet: Example

The noun `bar` has 11 senses:

- `Barroom, bar, saloon, ...` (a room where drinks are served)
- `Bar` (counter where you can purchase food or drink)
- `bar` (unit of pressure)
- `...`

The verb `bar` has 4 senses:

- `Barricade, block,...`
- `Bar, debar, exclude,...`
- `...`

WordNet: Structure

Parent hierarchy of the different meanings of `bar`:

- `Barroom, bar → room → area → structure → artifact ...`
- `Bar → counter → table → furniture → ... → artifact ...`
- `Bar → implement → instrumentation → artifact ...`
- `Bar → musical notation → notation → writing → ...`

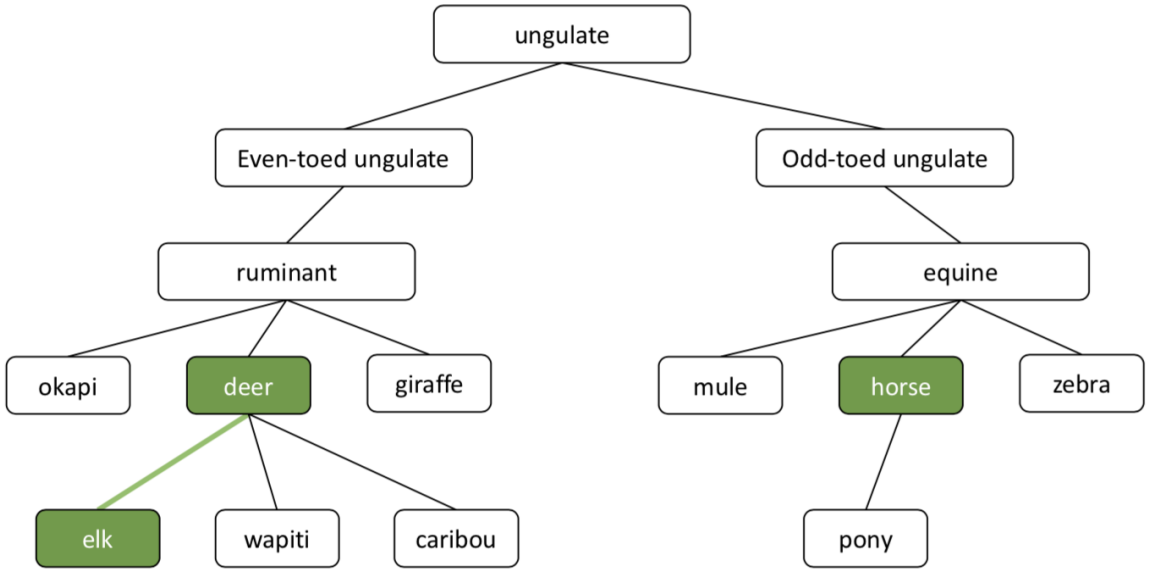
How is this helpful for computing word similarities?

However:

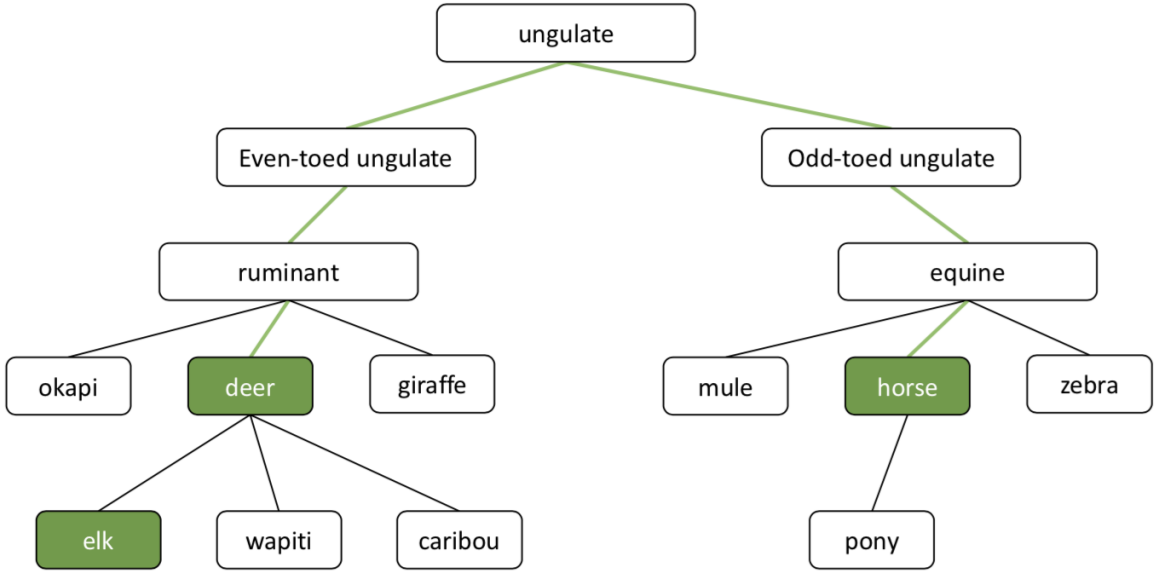
- Not all words share the same root
- WordNet has multiple roots
- WordNet is a forest, not just a tree

WordNet: Tree-based Similarity

Which pair is more similar? `Deer - elk` or `deer - horse`?



`TreeDist(deer, elk) = 1`



`TreeDist(deer, horse) = 6`

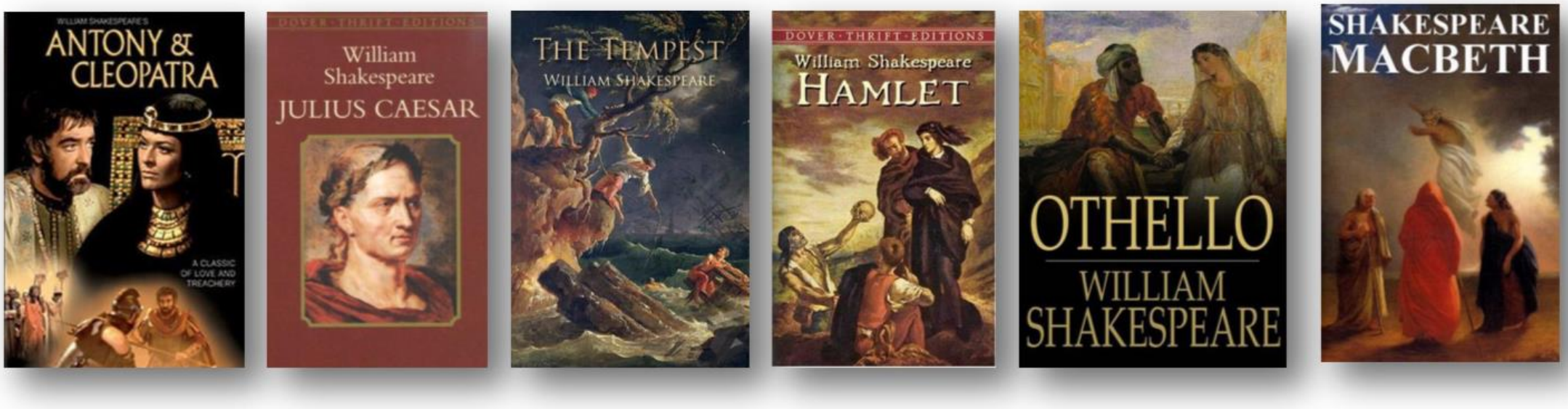
In principle, the higher the distance the lower the similarity! But there are problems:

- A specific word may not be in any tree
- Hypernymy edges are not all equally apart in similarity space
- Many more detailed graph-based semantic similarity measures have been developed.

The Vector Space Model

Comparing and Querying Documents

So far, we have discussed similarity between words. But what about documents?



For example, if we have a corpus with all plays by Shakespeare, how can we identify all plays that contain the words `Brutus` and `Caesar`?

→ How can we search?

Boolean Retrieval

The simplest information retrieval system:

- Create an index of all words in the documents
- Model queries as Boolean expressions (`Caesar AND Brutus`)
- The retrieval engine returns all documents that match the expression
- Stemming and lemmatization can help to improve recall

Term-Document Incidence Matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Mcbeth	…
Antony	1	1	0	0	0	1	…
Brutus	1	1	0	1	0	0	…
Caesar	1	1	0	1	1	0	…
Calpurnia	0	1	0	0	0	0	…
Cleopatria	1	0	0	0	0	0	…
mercy	1	0	1	1	1	1	…
worser	1	0	1	1	1	1	…

Indexing and retrieval:

- Create a matrix of the document collection that contains all distinct terms
- Set the value to `1` if the corresponding document contains the given term
- Return all documents that have a value of `1` in all cells corresponding to query terms

Scoring for Ranked Retrieval

But what happens if many documents match?

→ We want to return the documents in an order that is likely to be useful to the searcher

How can we rank (= order) the documents in the collection with respect to a query?

- Assign a score to each document (typically in the range `[0,1]`)
- The score measures how well the document and the query "match"
- We need a way to assign a score to a query/document pair

Term Frequency Matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Mcbeth	…
Antony	157	73	0	0	0	1	…
Brutus	4	157	0	2	0	0	…
Caesar	232	227	0	2	1	0	…
Calpurnia	0	10	0	0	0	0	…
Cleopatria	57	0	0	0	0	0	…
mercy	2	0	3	8	5	8	…
worser	2	0	1	1	1	5	…

To have data for a scoring function, term frequency information is helpful:

- We can store term frequency counts instead of binary values in the matrix
- Each document is now represented by a count vector
- Note: word order is not retained in a vector. This approach is called bag of words

Term Frequency (TF)

The term frequency $tf_{t,d}$ of term t in document d is defined as the number of times that t occurs in d.

- A document with 10 occurrences of the term `Hamlet` is more relevant for a query containing `Hamlet` than a document with just one occurrence
- But probably not 10 times more relevant

  Relevance does not increase proportionally with term frequency

⇨

How can we correct for this?

Rarity of Terms

Query example: `Are spider enthusiasts arachnocentric?`

Rare terms:

- Consider a term in the query that is rare in the corpus (e.g., `arachnocentric`)
- A document containing this term is very likely to be relevant

⇨ We want large positive weights for rare terms.

Frequent terms:

- Consider a term in the query that is frequent in the corpus (e.g., `spider`)
- A document containing this term is more likely to be relevant than a document that does not, but it is less of an indicator of relevance

⇨ We want positive weights for frequent terms (but lower than for rare terms)

Inverse Document Frequency (IDF)

The document frequency df_t of term t is defined as the number of documents in the corpus in which t occurs.

- We can use df_t to account for the rarity of t when computing the matching score
- The document frequency is an inverse measure of the informativeness of a term

⇨ So we need to invert it

The inverse document frequency idf_t is a measure of the informativeness of the term. We define the idf_t weight of term t as:

$$idf_t = \log_{10} \frac{N}{df_t}$$

Where N is the number of documents. We use $\log N/df_t$ instead of N/df_t as a heuristic to "dampen" the effect of the inverse document frequency.

Term Frequency-Inverse Document Frequency (TF-IDF)

We combine the term frequency and the inverse document frequency to assign a tf-idf weight w to each term t in each document d:

$$w_{t,d} = (1+\log_{10} tf_{t,d}) \log_{10} \frac{N}{df_t}$$

The weight:

- Increases with the number of occurrences of a term within a document
- Increases with the rarity of the term in the collection

Note: we use log-scaling of term frequencies and assume $tf_{t,d} > 0$ (otherwise, we set $w_{t,d} = 0$). Other approaches are possible and there is no rigorous formal reasoning behind this choice. In practice, experiments help to determine suitable scaling methods.

TF-IDF Weight Matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Mcbeth	…
Antony	5.25	3.18	0	0	0	0.35	…
Brutus	1.21	6.10	0	1.0	0	0	…
Caesar	8.59	2.54	0	1.51	0.25	0	…
Calpurnia	0	1.54	0	0	0	0	…
Cleopatria	2.85	0	0	0	0	0	…
mercy	1.51	0	1.90	0.12	5.25	0.88	…
worser	1.37	0	0.11	4.15	0.25	1.95	…

Each document is now represented by a real-valued vector of tf-idf weights

- Weights encode frequency information of terms in documents
- Frequencies of terms in a documents are normalized by the number of documents in which the term occurs

Document Similarity

Documents as Vectors

Each document is now represented by a real-valued vector of tf-idf weights.

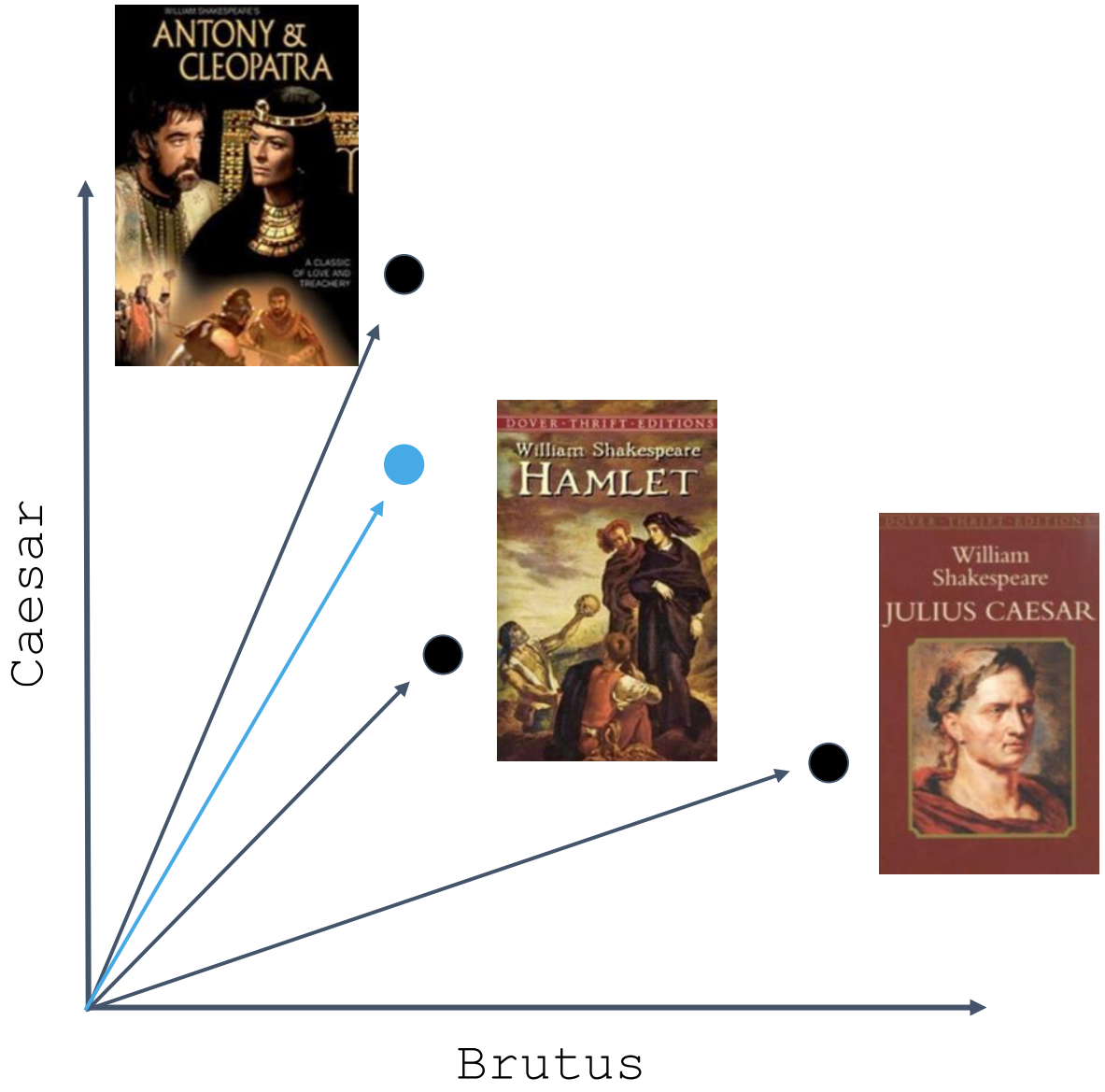
- Terms are dimensions (= axes) of the corresponding vector space
- Documents are points or vectors in this space
- The vector space is very high-dimensional due to the vocabulary size:
  Tens of millions of dimensions when building search engine at web scale
- Individual vectors are very sparse: most entries are zero

How is this helpful in querying?

- Idea: Find documents vectors that are similar to the query vector in this space

⇨ Finding close points in a vector space is a well-researched problem

Documents in Vector Space



	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet
Antony	5.25	3.18	0	0
Brutus	1.21	6.10	0	1.0
Caesar	8.59	2.54	0	1.51
Calpurnia	0	1.54	0	0
Cleopatria	2.85	0	0	0
mercy	1.51	0	1.90	0.12
worser	1.37	0	0.11	4.15

Query:

Brutus killed Caesar

Measuring Distances in Vector Space

Intuitively: Euclidean distance between query vector and and document vector.

However, this is a bad idea...

- The Euclidean distance between query and document will be large
- This is true even if the distribution of terms in the query and the

⇨ distribution

of terms in the document are very similar

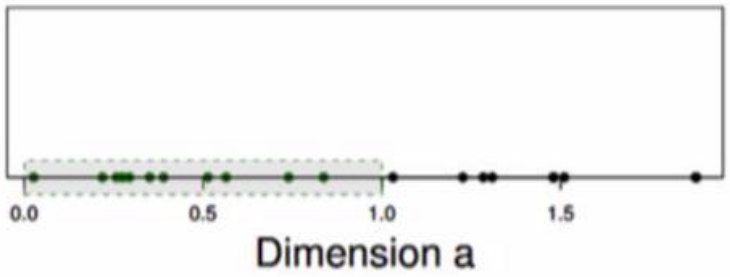
- Reason: Euclidean distance is large for vectors of different lengths

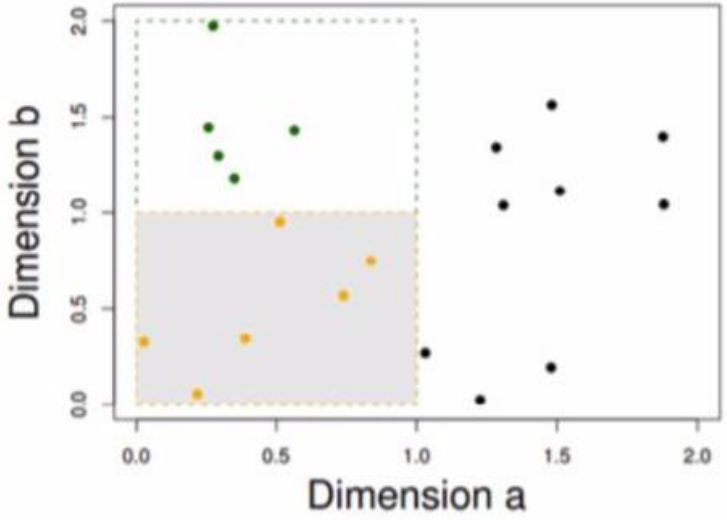
Curse of dimensionality

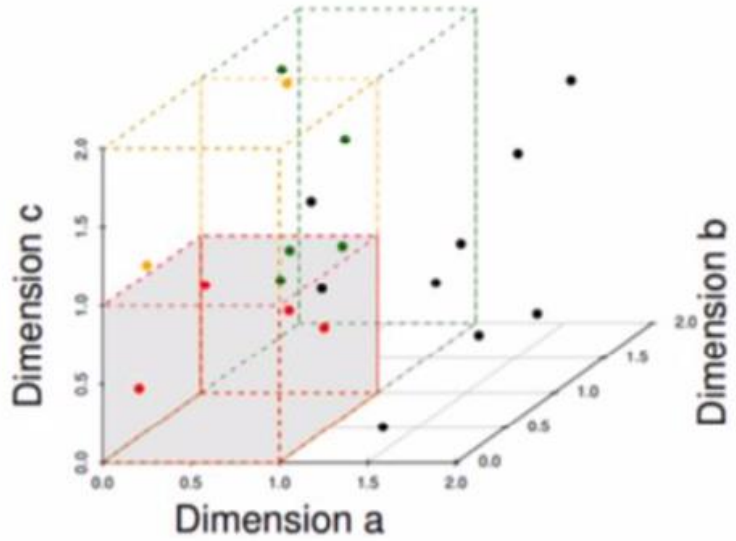
Curse of Dimensionality

Problem:

- When the dimensionality increases, the volume of the space increases exponentially







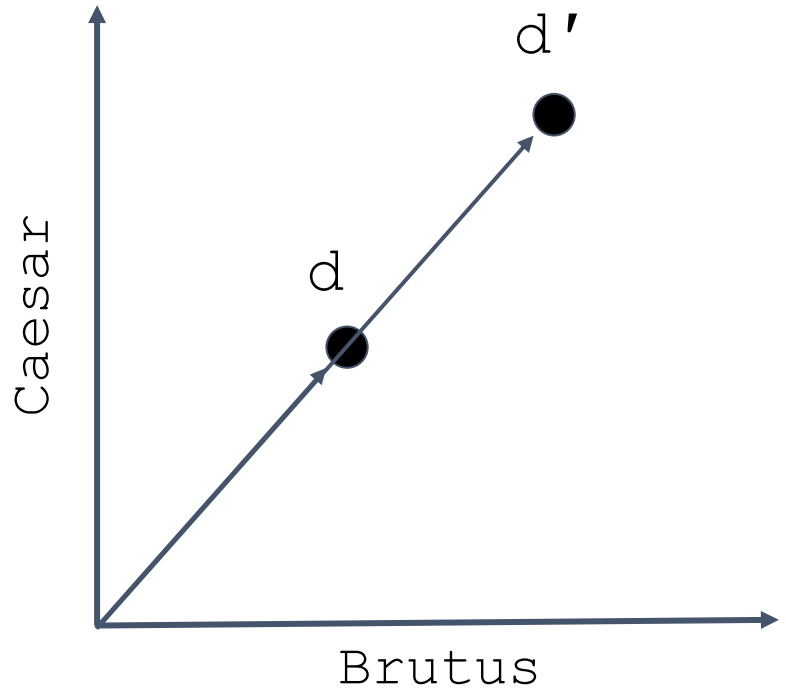
Using Angular Similarity

Solution:

- Rank documents according to their angular distance from the query

Thought experiment:

- Take a document d and append it to itself.
  Call this document d′
- "Semantically" d and d′ have exactly the same content
- The angle between the two documents is 0°,
  corresponding to maximal similarity
- The Euclidean distance between d and d′
  scales with the number of tokens in document d



In a Nutshell: A Simple Search Engine

We can now construct a simple (yet surprisingly effective) search engine:

- Compute tf-idf weights for all terms in all documents in the corpus
- Represent the corpus as a tf-idf weighted incidence matrix
- Compute a tf-idf weighted vector representations of the query
- Compute the cosine distance between the query vector and all document vectors
- Rank the documents non-decreasingly by cosine distance score
- Return the top 10 results as the 10 blue links

Online Resources

Levenshtein Distance demo

https://phiresky.github.io/levenshtein-demo/

WordNet

https://wordnet.princeton.edu/

Further-Watching Material

Minimum Edit Distance

https://youtu.be/jhSdB36RYWk?list=PLoROMvodv4rOFZnDyrlW3-nI7tMLtmiJZ

https://youtu.be/Q7QQCNM7AJ4?list=PLoROMvodv4rOFZnDyrlW3-nI7tMLtmiJZ

Semantic Similarity

https://youtu.be/PxgkddPbjrM

Term-Document Incidence Matrix

https://youtu.be/b6cjKqTE04s?list=PLoROMvodv4rOFZnDyrlW3-nI7tMLtmiJZ

Vector Space Model

https://youtu.be/yvNt_qDRbDQ?list=PLoROMvodv4rOFZnDyrlW3-nI7tMLtmiJZ

Language Models

Applications of Language Models

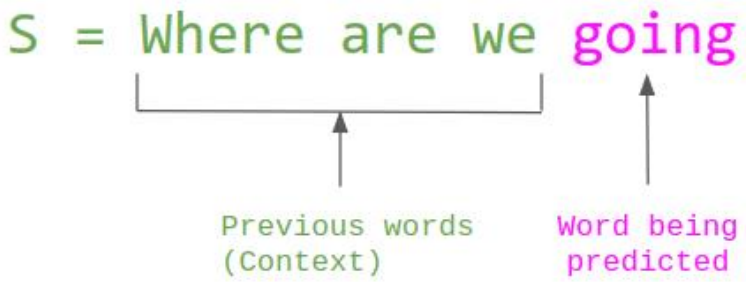
- Natural language understanding (NLU)
  - Question answering systems
  - Chatbots
- Natural language generation (NLG)
  - Text summarization
  - Automated journalism
  - Chatbots
- Machine translation
- Text classification
- Named entity recognition
- Opinion mining
- Sentiment analysis
- ...

Language Models: Definition

Language model:

A language model is a probability distribution over sequences of words.

By using a language model, we can assign probabilities to words, given a sequence of other words. That is, we



$$P(S) = P(\text{Where}) \times P(\text{are} \mid \text{Where}) \times P(\text{we} \mid \text{Where are}) \times P(\text{going} \mid \text{Where are we})$$

Language model examples:

- n-grams
- Static embeddings
  - Word2vec
  - GloVe
  - FastText
  - etc.
- Contextualized word embeddings
  - GPT-2 / GPT-3
  - BERT
  - WuDao 2.0
  - T5
  - etc.

A Core NLP Task: Word Similarity

Many applications in text processing and information retrieval rely on word similarity as a core task that needs to be solved.

- Spell checking
  - Similarity between individual words
- Search
  - Similarity between a sentence and the content of a document
- Duplicate detection
  - Similarity between two documents
- Summarization
  - Removal of redundant (= similar) sentences in a document
- Translation
  - Finding a similar word in a different language

A similarity between sentences or documents can often be derived from word similarities.

Word Similarity in Manually Created Models

We already know that manually curated models of language like WordNet are a great resource that can be used to compute word similarities. But:

- WordNet is missing nuance
  - For example, `proficient` is a synonym for `good`.
  - `Proficient cop, bad cop` is different from `Good cop, bad cop`.
  - This type of similarity is only correct in some contexts.
- WordNet is missing new meanings of words until they are added
  - `Streaming` (on Twitch), `salty` (= angry), etc.
  - Impossible to keep manually curated resources up-to-date
- WordNet is subjective and biased by the annotators' perspectives
- Requires constant human labor to create and update

Similarity in the Vector Space Model

We also know how to compute similarities between a query sentence and a document in the vector space model:

- Represent the query as a tf-idf encoded vector
- Represent each document as a tf-idf encoded vector
- Compute cosine similarities between query and documents

Can we use this to handle word similarities?

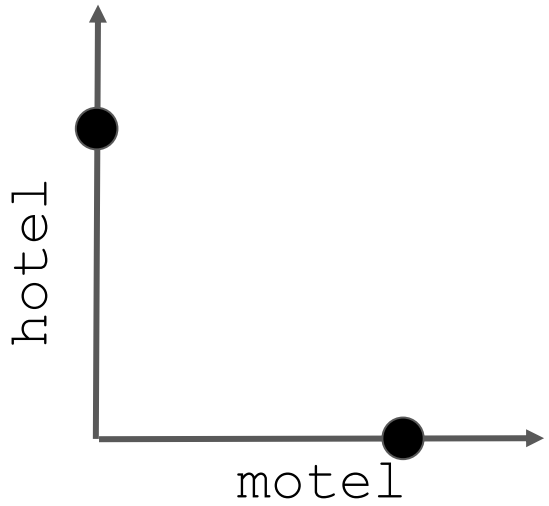
Not really…

- Word 1: `Hotel`, w_1 = `[0, 0, 0, 0, 1, 0, 0]`
- Word 2: `Motel`, w_2 = `[0, 0, 0, 1, 0, 0, 0]`
- `CosineSim(`w_1`,`w_2`)` = 0
- Normalizing the values of w_1 and w_2 with tf-idf does not change the similarity.

One-Hot Vector Encodings

In the vector space model, we are using a localist representation:

- The dimension of the vector space is equal to the vocabulary size
- Each word (or lemma, or stem) corresponds to exactly one dimension
- A single word is encoded by a one-hot vector:
  - All vector components are equal to 0 (these components are “cold”)
  - Only the component in the dimension that corresponds to the word itself has a value of 1 (the component is “hot”)
- Individual word vectors are orthogonal by definition
- There is no notion of word similarity in the vector space model



Fundamental Limitations of the Vector Space Model

The vector space model is designed to compare sequences of text based on the words they contain. But even that is problematic. Consider the search queries:

- Query: `Konstanz Motel` q_1 = [0, 1, 0, 0, 1, 0, 0]
- Doc 1: `...Konstanz Hotel...` d_1 = [0, 1, 0, 1, 0, 0, 0]
- Doc 2: `...Universität Konstanz...` d_2 = [0, 1, 0, 0, 0, 0, 1]

A traveler from the U.S. who is searching for a motel in Konstanz is equally likely to find hotel websites as they are to find websites related to the university.

Historical solution in information retrieval: Query expansion

- Keep a dictionary of similar words and use it to expand the queries
- E.g., if a user searches for `motel`, also add `hotel` to the query

Towards Semantically Meaningful Word Representations

Query expansion is a (set of) heuristic(s) specifically designed to work around the inability of the vector space model to represent semantic meaning.

For handling word similarities, the vector space model won't do! Can we do better?

Ideas?

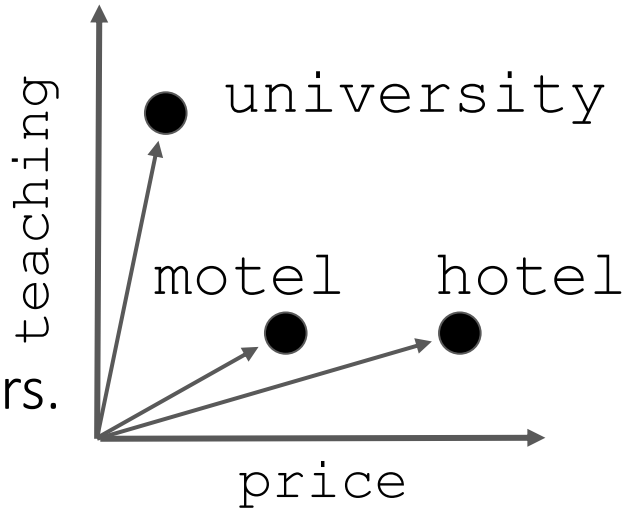
Dense Word Vector Representations

In theory:
Instead of a term-document matrix, we want a term-concept matrix, in which each word is ranked according to the strength of its relation to a semantic concept. For example:

	price	teaching
hotel	0.9	0.1
motel	0.5	0.1
university	0.1	0.8

- `Hotel` and `motel` are very similar and differ mostly in room prices and the quality of service.
- They are both similarly low on a scale that rates the relevance for teaching, while a `university` is higher on that scale.

The rows of such a matrix could then be used a vector repre-sentations of the words. Similar words would have similar vectors.

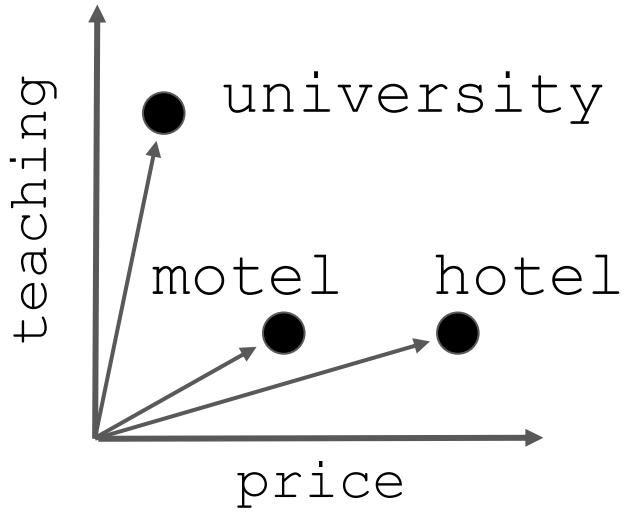


Dense Word Vector Representations

In practice:
How can we even create such a matrix? Manual curation is entirely infeasible:

- How do we decide on the values?
- How do we choose the concepts?
- We still have to update the model with new words
- We also have to update in case of semantic shifts (e.g., `gay` in the 1930s vs. today)

	price	teaching
hotel	0.9	0.1
motel	0.5	0.1
university	0.1	0.8



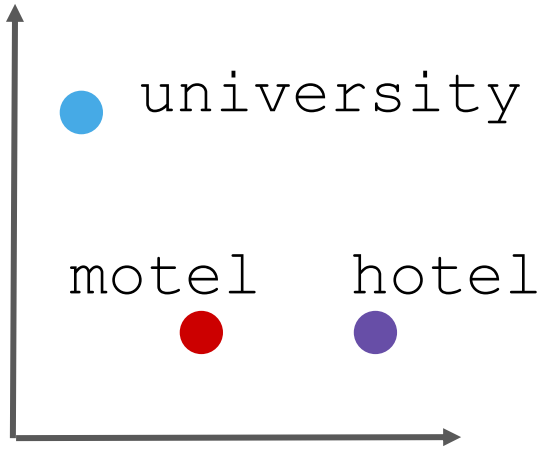
⇨ We need a way of learning dense vector representations from data.

Word Embeddings

Word Embeddings: Definition

Embedding of words:
Given a training corpus, the embedding of words is the process of assigning a vector (in a latent space) to each word, based on the content of the corpus (by using some algorithm).

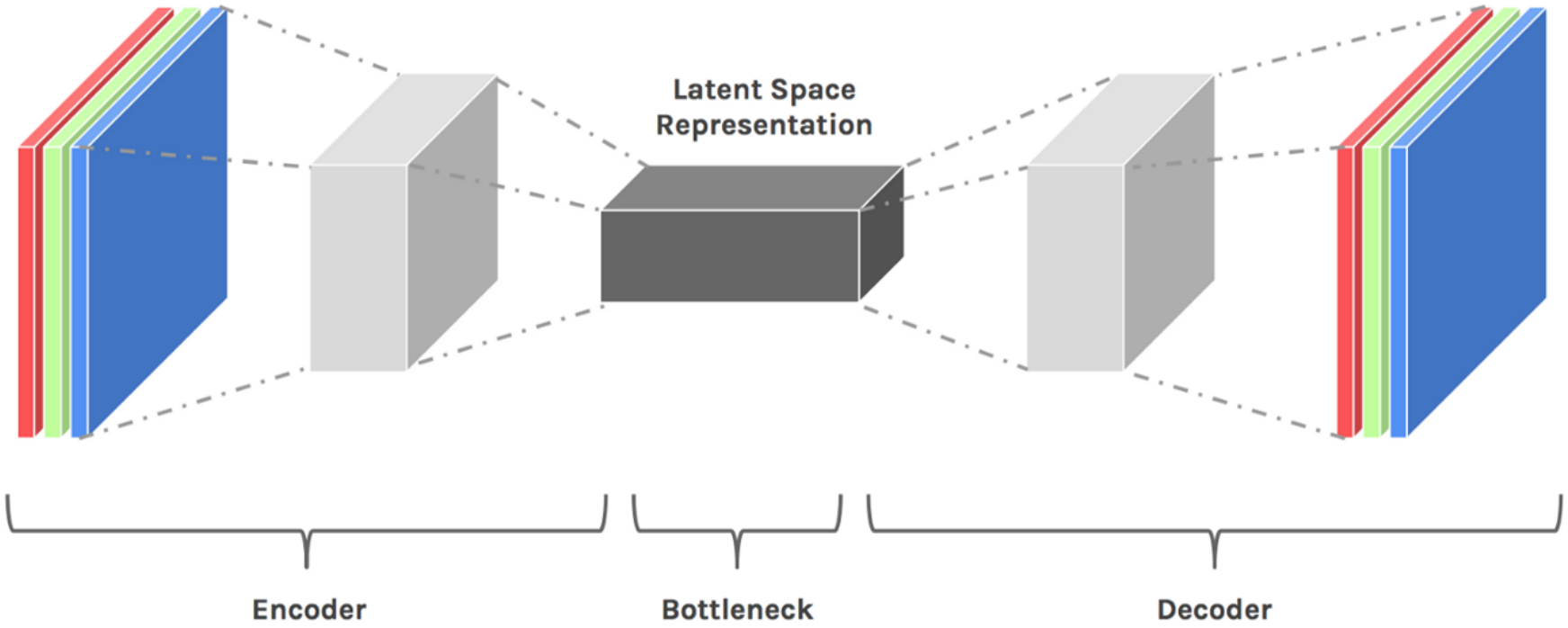
Word embedding:
The vector that is assigned to a word as the result of the above process is often called the embedding of the word.



Konstanz is a German city that has a university and several hotels, but no motels.

Latent Space

A latent space (also called embedding space) is a vector space in which items are placed in such a way that similar items are placed in proximity. Typically (but not always), latent spaces have a lower rank (dimensionality) than the original space of the data.



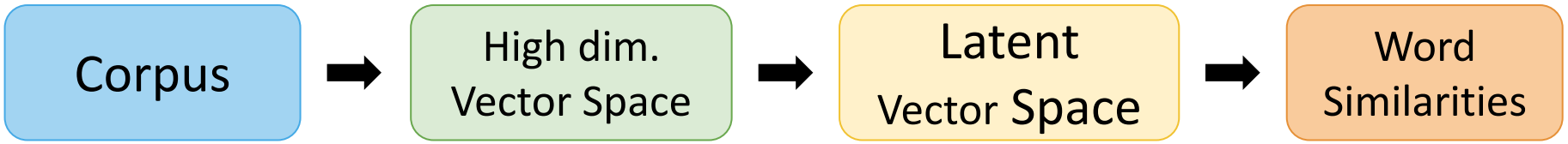
https://towardsdatascience.com/understanding-latent-space-in-machine-learning-de5a7c687d8d

Embedding Spaces for Words

For creating word embeddings, the embedding algorithm should:

- Place similar and/or related words in close proximity in the latent space
- Use word (co-)occurrence information from the corpus
- Work without supervision (labeling data at this this scale is infeasible)

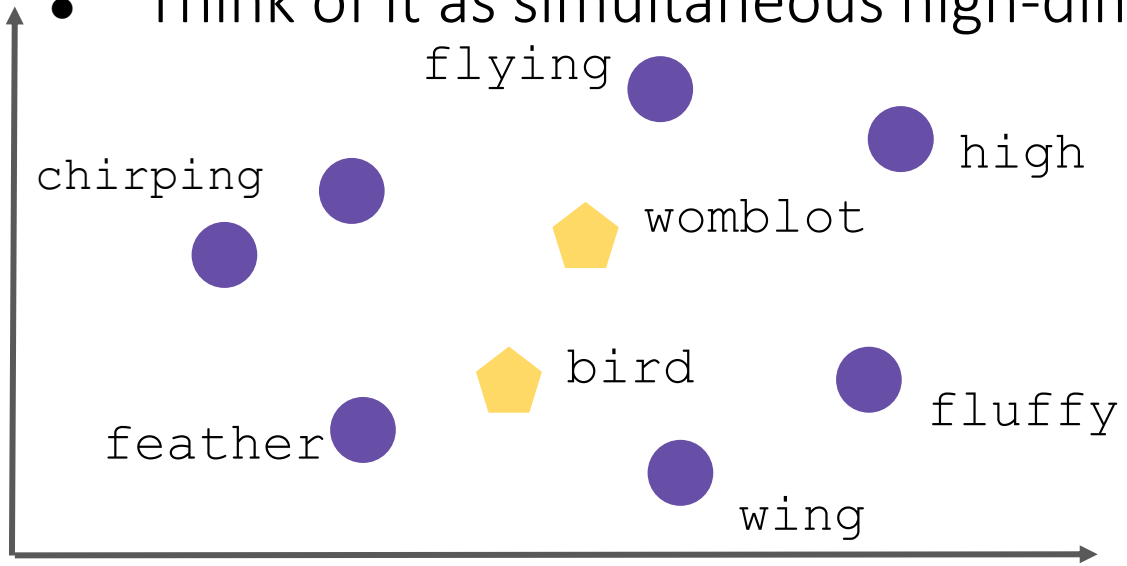
Embedding Pipeline:



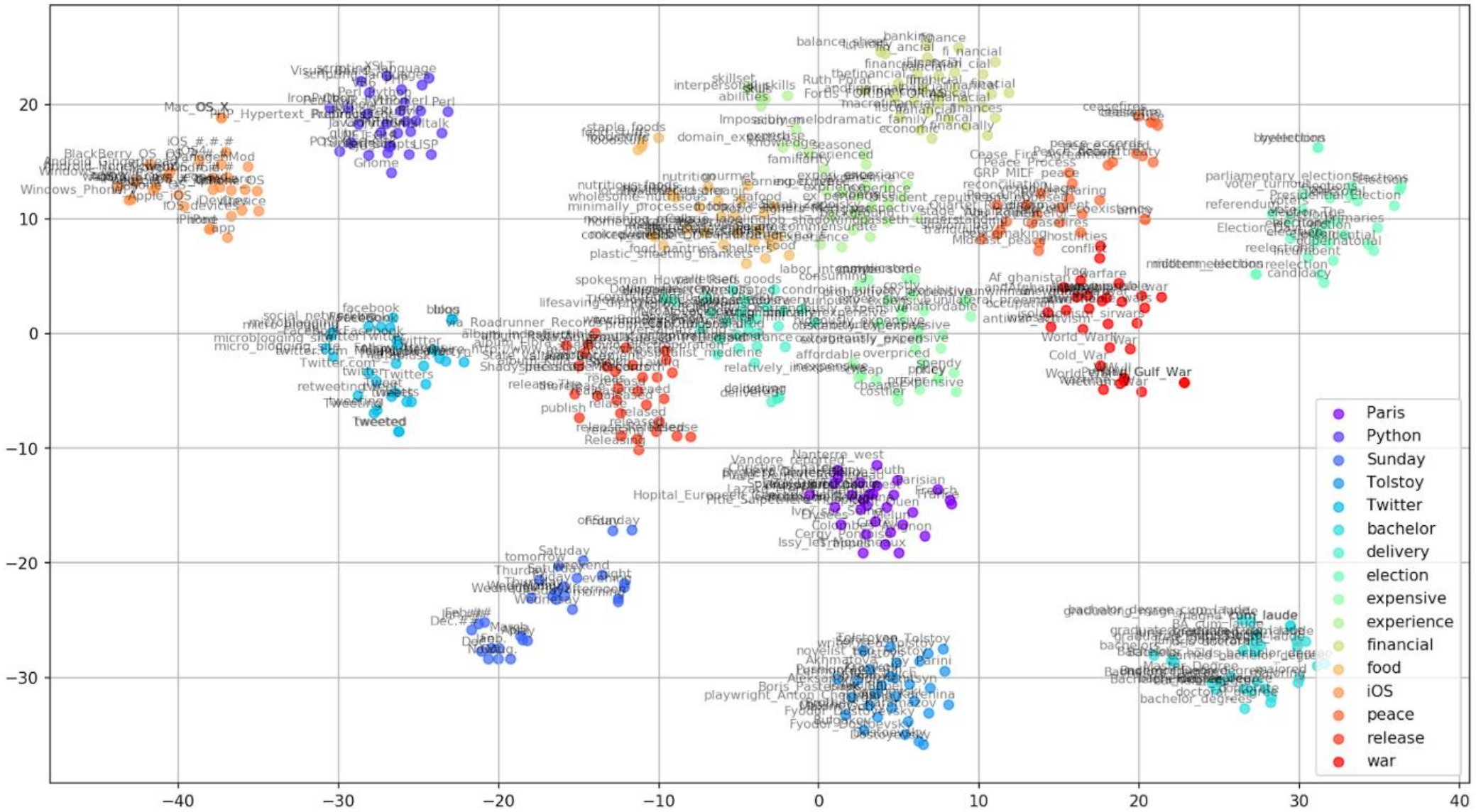
How word2vec Works Intuitively

Why does word2vec produce word embeddings that produce meaningful word similarity?

- During initialization, each word starts at a random location in the latent space
- During training, each word is pulled closer towards frequently cooccurring words
- During training, each word is pushed away from non-cooccurring words
- Words end up in the proximity of other words that occur in similar contexts
- Think of it as simultaneous high-dimensional tug-of-war



Dense Word Embeddings: Visualized



https://towardsdatascience.com/google-news-and-leo-tolstoy-visualizing-word2vec-word-embeddings-with-t-sne-11558d8bd4d

Whats your current status?

Which data did you choose?

What is your user domain?

What is your task?

What are NLP related problems?