

# Information Management: Natural Language Processing

## Summary

Fabian Damken

November 8, 2023



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

---

# Contents

---

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Language and Knowledge Processing . . . . .	4
1.1.1	Implicit Information . . . . .	4
1.1.2	Text- and Data-Mining . . . . .	4
1.1.3	Examples . . . . .	5
1.2	Major Scientific Challenges . . . . .	6
1.2.1	Ambiguity . . . . .	6
1.2.2	Underspecification and Vagueness . . . . .	7
1.2.3	Scientific Data . . . . .	7
1.2.4	System Evaluation . . . . .	7
<b>2</b>	<b>Natural Language Text</b>	<b>8</b>
2.1	Scripts and Language . . . . .	8
2.2	Properties of Text . . . . .	8
2.2.1	Structure . . . . .	9
2.3	Examples of electronic Text Formats . . . . .	9
2.3.1	Text in relational Databases . . . . .	9
2.4	Character Encoding . . . . .	10
2.4.1	Encoding Issues in Software and Databases . . . . .	10
2.4.2	Example Encodings . . . . .	11
<b>3</b>	<b>Linguistic Preprocessing</b>	<b>14</b>
3.1	Segmentation . . . . .	14
3.1.1	Tokenization . . . . .	14
3.1.2	Segmentation in other Languages . . . . .	15
3.1.3	Software . . . . .	15
3.2	Morphology . . . . .	15
3.2.1	Bases and Affixes . . . . .	16
3.2.2	Classification of Morphemes . . . . .	16
3.2.3	Word Formation . . . . .	17
3.2.4	Morphology in other Languages . . . . .	17
3.2.5	Morphological Normalization . . . . .	17
3.2.6	Software . . . . .	19
3.3	Syntax . . . . .	19
3.3.1	Part of Speech Tagging (POS Tagging) . . . . .	19
3.3.2	Parsing . . . . .	21
3.4	Semantic . . . . .	26
3.4.1	Ambiguity . . . . .	26
3.4.2	Word Sense Disambiguation . . . . .	27

<b>4</b>	<b>Text Corpora, Lexical Resources and Knowledge Bases</b>	<b>28</b>
4.1	Text Corpora . . . . .	28
4.1.1	Parameters of a Corpus . . . . .	28
4.1.2	Content of a Corpus . . . . .	29
4.1.3	Distributional Hypothesis . . . . .	29
4.1.4	Corpus Creation . . . . .	30
4.1.5	Corpus Query Languages . . . . .	30
4.1.6	Example Corpora . . . . .	30
4.2	Lexical Resources and Knowledge Bases . . . . .	31
4.2.1	Lexical Resources ↔ Corpora . . . . .	32
4.2.2	Meaning, Lexical Ambiguity, Synonymy . . . . .	32
4.2.3	Relation Types . . . . .	32
4.2.4	Wordnets . . . . .	32
<b>5</b>	<b>Information Retrieval</b>	<b>33</b>
5.1	Overview . . . . .	33
5.1.1	Basic Concepts . . . . .	33
5.1.2	IR Engine vs. DBMS . . . . .	33
5.1.3	Core Challenges in IR . . . . .	33
5.1.4	IR Activities . . . . .	34
5.2	Boolean Retrieval . . . . .	34
5.2.1	Term-Document Matrix . . . . .	34
5.3	Vector Space Model . . . . .	35
5.3.1	Boolean Search Issues . . . . .	35
5.3.2	Ranked Retrieval . . . . .	36
5.3.3	Set of Words, Bag of Words . . . . .	37
5.3.4	Vector Space Model (VSM) . . . . .	37
5.3.5	Search the VSM . . . . .	38
5.3.6	Vector Weights . . . . .	39
5.3.7	VSM Retrieval: Example . . . . .	40
5.4	Information Retrieval Evaluation . . . . .	40
5.4.1	$F_1$ Score . . . . .	41
5.4.2	Cutoff and Precision at Rank . . . . .	41
<b>6</b>	<b>Information Extraction and Classification</b>	<b>42</b>
6.1	Information Extraction (IE) . . . . .	42
6.1.1	Information Retrieval vs. Information Extraction . . . . .	42
6.2	Entity Recognition . . . . .	42
6.2.1	Entity Types . . . . .	42
6.2.2	Challenges . . . . .	43
6.2.3	Approaches . . . . .	43
6.2.4	Next Steps After Information Extraction . . . . .	44
6.3	Supervised vs. Probabilistic Machine Learning . . . . .	45
6.3.1	Classification . . . . .	45
6.3.2	General (Supervised) ML Workflow . . . . .	46

---

# 1 Introduction

---

Natural Language Processing is all about:

- Interpreting (creating information)
- Linking (supporting knowledge acquisition)
- Analyzing
- Verifying

---

## 1.1 Language and Knowledge Processing

---

- The World Wide Web grows about by 25 PB per day.
- Most of this data is unstructured.

---

### 1.1.1 Implicit Information

---

- Data always contains *implicit data*.
- Example: Nearly every supermarket stores information about each transaction. This data includes lots of implicit information:
  - products rarely sold
  - products often bought together
  - ...
- Language and knowledge processing is about making this implicit information explicit.

---

### 1.1.2 Text- and Data-Mining

---

- **Data Mining:** Exploring and analyzing large amount of data automatically to discover meaningful patterns.
  - Machine learning
  - Distributed systems, Databases
  - Data modeling and pattern recognition
  - Software engineering
- **Text Mining:** Exploring and analyzing large amount of *text* automatically to discover meaningful patterns.
  - Natural Language Processing.

---

### 1.1.3 Examples

---

#### Recommender Systems

---

**Given** A set of items and background information about users, items, . . . .

**Goal** Present the user with a subset of items that use user will like.

---

#### Machine Learning (ML)

---

**Given** Task, experience, performance feedback

**Goal** Perform a task well, improving it with experience

- A fundamental method for many text and data mining tasks.
- 

#### Document Classification

---

**Given** Document, set of Topics

**Goal** Find a topic in the given set of topics that describes the given document.

---

#### Machine Translation (MT)

---

**Given** A text in a language.

**Goal** Output a text in another language with the same meaning as the given text.

---

#### Speech Processing

---

##### Automatic Speech Recognition

**Given** Recorded voice.

**Goal** Find the transcript of the given recording.

##### Speech Synthesis

**Given** Written text (transcript).

**Goal** Find the corresponding acoustic signals for the given text.

---

#### Information Retrieval (IR)

---

**Given** A document collection and a query.

**Goal** Find a subset of the documents which is maximally relevant to the query.

---

---

## Information Extraction (IE)

---

**Given** A document collection.

**Goal** Extract structured knowledge (like entities, events, relationships, ...).

---

## Question Answering

---

**Given** A natural language question and knowledge sources.

**Goal** Generate an answer for the question based on the knowledge sources.

---

## Automatic Summarization

---

**Given** A document collection.

**Goal** Find a smaller text that summarizes the documents.

- Variations
    - Single- vs. Multi-document summarization
    - Abstractive vs. Extractive summarization
    - Indicative vs. Informative summarization
    - General vs. Query-focused summarization
- 

## Sentiment Analysis and Opinion-Mining

---

**Given** A text.

**Goal** Separate subjective/objective opinions from objective statements; distinguish positive/negative statements; identify specific opinions and opinion targets

---

## Intelligent Computer-Assisted Language Learning (ICALL)

---

**Given** Learning and a learning goal.

**Goal** Assist the learning with intelligent tools.

---

## 1.2 Major Scientific Challenges

---

### 1.2.1 Ambiguity

---

See chapter 3.

The sentence “**pretty** little girl’s school” can refer to:

- Very small girl going to school.
  - Beautiful girl going to school.
  - Very small school the girl goes to.
  - Beautiful school the girl goes to.
-

---

## 1.2.2 Underspecification and Vagueness

---

“Alice is pretty little.”

- $\text{height}(\text{Alice}, x)$
- $x = 0 \vee x = 3$  is very unlikely.
- $x = 1.6m$  is more likely.

“He usually does other things before dinner.”

- Who is “he”?
- What are these “other things”?
- What does “before dinner” mean? One hour or ten days? When is dinner time?
- ...

---

## 1.2.3 Scientific Data

---

See chapter 4.

Most of the NLP methods require data for

- training,
- development and
- evaluation.

It is often hard and expensive to obtain high-quality data, for both researchers and companies.

For example: Find a data set with more than a million examples for good/bad arguments.

---

## 1.2.4 System Evaluation

---

- Intrinsic evaluation
  - Test the method in isolation by comparing the actual output with the expected output (the so-called gold standard).
- Extrinsic evaluation
  - Test the method in use by embedding it into a larger system or experimental setup (e.g. a user study).

---

## 2 Natural Language Text

---

**Alphabet**  $\Sigma = \{A, B, C, \dots, a, b, c, \dots, 0, 1, \dots\}$

**Characters**  $s' \in \Sigma$

**String**  $s = s_1 \dots s_n \in \Sigma^*$

**Message** A string that follows a grammar.

**Data** An exchanged message that is not yet interpreted.

**Text** Written coherent data of a language.

**Coherence** Coherence is what makes a text semantically meaningful and connects the ideas arranged in a text.

---

### 2.1 Scripts and Language

---

- Script
  - Defines the symbol set  $\Sigma$ .
  - *Alphabetic* scripts: Latin, Greek, Cyrillic, ...
  - *Logographic* or *syllabic*: Hanzi (Chinese), Kanji, Katakana, ...
  - Consonant-Based (*Abjab*): Arabic, Hebrew, ...
  - Segment-based (*Abugida*): Devanagari, Tibetan, Indic, ...
  - Writing direction: left-to-right, right-to-left, top-to-bottom, bottom-to-top, ...
- Language
  - Determines the grammar and the vocabulary.
  - Examples: English, German, Japanese, Spain, ...

---

### 2.2 Properties of Text

---

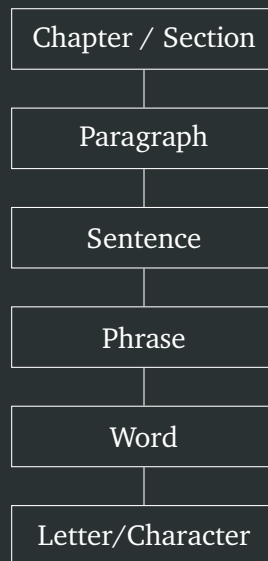
- **Text type/Genre**: textbook, scientific paper, news item, social media post, ...
- **Register**: language variety (dialect, jargon, formal, ...)
- **Style**: how the text is written (e.g. funny, political, scientific, ...)
- **Domain/topic**: biology, information science, education, spacecrafts, ...
- **Time** of writing as languages change during time



---

### 2.2.1 Structure

---



---

## 2.3 Examples of electronic Text Formats

---

- Mostly unstructured
  - Plain text
- Semi-structured
  - Hypertext markup language (HTML)
  - Extensible markup language (XML)
- XML-based formats
  - Open document format (ODT)
  - Word files (DOCX)
- Binary content
  - Word files (DOC)
- Mixed text and binary content
  - Rich text (RTF)
  - Portable document format (PDF)

---

### 2.3.1 Text in relational Databases

---

- It is also possible to store large amount data in a relational database.
- `CHAR(n)` and `VARCHAR(n)` can only store up to *n* characters with typically  $n \leq 4000 \cdots 8000$ .
- A lot more text with more or less arbitrary length can be saved using:

- Binary large objects (BLOBs)
- Character large objects (CLOBs)
- The size limit for these data types are mostly infinite with ranging from 2 GB to more than 128 TB.
- But in most DBs it is not possible to do sorting/group and comparing is limited to the most basic (=, BETWEEN, ...).

---

## 2.4 Character Encoding

---

- As databases can only store 1s and 0s a character encoding is needed:

$$\text{enc} : \Sigma^* \rightarrow \{0, 1\}^* \quad (\text{Encoding})$$
$$\text{dec} : \{0, 1\}^* \rightarrow \Sigma^* \quad (\text{Decoding})$$

- Typically each character is mapped onto a bit mask using a character set  $\Psi : \Sigma \rightarrow \{0, 1\}^*$ , so a text  $T = c_1 \cdots c_n \in \Sigma^*$  is encoding as  $\text{enc}(T) = \Psi(c_1) \cdots \Psi(c_n)$
- There are several base categories of encoding:
  - **Single byte**  
Each character corresponds to a single byte (8 Bit).
  - **Multi byte**  
Each character corresponds to a fixed number of bytes.
  - **Variable-length**  
The bit length of each character might change depending on the character.
  - **Escape-code-based**  
Switch between different character sets using escape codes.

---

### 2.4.1 Encoding Issues in Software and Databases

---

- Key question when working with text: Which encoding does the text use?
- Key question when programming: Default encoding of source files? Default encoding of string types? Default encoding when reading/writing a file?
- Key questions when working with databases: Default encoding of a table? Default encoding of the database connection?
- Example Issues
  - String length of variable-length encodings
  - String operations
  - String searching
  - Sort strings alphabetically

---

## Sorting

---

- **Order of Characters**
  - Latin script:  $A < B < \dots < Y < Z$
  - Chinese: How to sort? By number of strokes?
- **Case**
  - Binary sort:  $A < B < \dots < a < b < \dots$
  - Alternatives:  $A = a < B$  vs.  $A < a < B$
  - Turkish: `istanbul` = `iSTANBUL`
  - German: `istanbul` = `ISTANBUL`
- **Transliteration**
  - Greek:  $\alpha < \beta < \gamma < \delta$
  - German:  $a < b < g??? < d$
- **Equivalent Characters**
  - German (dictionary, DIN 5007-1):  $n < o = \ddot{o} < p$
  - Finnish:  $n < o < p < \dots < z < \ddot{a} < \ddot{o}$
- **Diacritics**
  - $e < \acute{e} < f$  vs.  $e = \acute{e} < f$
  - French: `côte` < `coté`
  - English: `coté` < `côte`
- **Character Combinations**
  - German (phonebook, DIN 5007-2):  $n < o [\ddot{o} = oe] < p$
  - Spanish (traditional):  $ce < cu < ch < da$
  - Danish:  $z < \text{æ} < \text{œ} = \text{ø} = \ddot{o} < \text{å}$

---

## Collation Framework

---

The collation...

- is the definition of a sort order,
- is available for multiple language, e.g. German and
- there are variants for sorting, e.g. lexical, phone book, traditional vs. modern, ...

---

### 2.4.2 Example Encodings

---

---

#### ASCII

---

See <https://www.asciitable.com>.

---

## Unicode

---

- The idea was to define one character set that covers all characters observed in the world including:
  - logographic languages
  - fantasy languages
  - currency symbols
  - emoji characters
  - ...
- Unicode 14.0 defines 144,697 characters, 159 writing systems and 3633 emojis.
- Each character has 4 bytes with currently 21 significant bits.

## Architecture

- Overall count of 1,114,112 code points ranging from 0x000000 to 0x10FFFF.
- The notation is U+ppxxxx.
- pp subdivides the code into 17 planes (from 0x00 to 0x10).
- The plane pp = 0 is called the *Basic Multilingual Plane* (BMP) and contains the most commonly used characters:
  - Latin, Greek, Cyrillic
  - Arabic, Hebrew
  - Most of the Chinese, Japanese and Korean languages (CJK)
  - ...
- Other planes are:
  - Plane 1** Supplementary Multilingual Plane (SMP)  
Historic languages
  - Plane 2** (Supplementary Ideographic Plane) (SIP)  
Extensions for Chinese, Japanese, Korean
  - Plane 3-13** Empty
  - Plane 14** Supplementary Special-purpose Plane (SSP)  
Control characters (e.g. language tags)
  - Plane 15-16** Supplementary Private Use Area (PUA-A/B)  
Reserved for private characters of a certain font

---

## UTF-32/UCS-4

---

- Each character is encoded with exactly 4 bytes.
- Directly corresponds to Unicode.
- Disadvantages:

- 
- Wastes disk space and memory, 4 times the size of ISO 8859-1.
  - This might yield to longer processing times.

---

## UCS-2

---

- Each character is encoded with exactly 2 bytes.
- Directly corresponds to the Unicode BMP.

---

## UTF-16

---

- Each character is encoded with 2 to 4 bytes.
- Directly corresponds to the Unicode BMP.
- Divides characters of supplementary planes into a so-called high and low surrogate.
- Surrogates: Subtract 0x10000 and divide into upper/lower 10 bits.

---

## UTF-8

---

- Each character is encoded with 1 to 4 bytes.
- Directly corresponds to ASCII.
- Non-ASCII characters use additional bytes.
- Theoretically it is possible to extend the code to 6 bytes.

---

## 3 Linguistic Preprocessing

---

### 3.1 Segmentation

---

The | dwarfs | loved | her | dearly

#### 3.1.1 Tokenization

---

- The task of *tokenization* is to segment an input stream into an ordered sequence of tokens.
- Most of the time, *tokens* correspond to inflected word forms.
- The system for segmenting the input stream is called *tokenizer*.
- Naive approach: split on whitespaces.
  - This leads to multiple problems, for instance with the following sentence:  
Mr. Sherwood said, reaction to Sea Containers' proposal has been "very positive."
  - Splitting on whitespaces gives the following false tokens:
    - \* said,
    - \* "very
    - \* positive."
  - Thus, splitting on whitespaces is not a solution.

---

#### Ambiguities

---

- Periods
  - In most cases, a period represents a sentence termination  $\implies$  separate tokens.
  - But sometimes, it is part of a token:
    - \* abbreviations (e.g. e.g.)
    - \* Ordinal numbers (e.g. 21.), fractions (e.g. 12.34)
    - \* references to resource locators (e.g. www.twitter.com)
- Comma
  - Part of numbers (e.g. 1,234)  $\implies$  part of token.
- Whitespace
  - Part of numbers (e.g. 1 234)  $\implies$  part of token.

- Single Quote
  - In some cases, it is an enclosing quote (e.g. "You asked 'why'," he said.)  $\Rightarrow$  separate tokens.
  - In contractions or elisions (e.g. don't, James' book)  $\Rightarrow$  should it be a part of the token or a separate token?
- Dash
  - In most cases, it is part of a token (e.g. part-time job).
  - But it may be part of ranges (e.g. see page 100-102)  $\Rightarrow$  may be separate tokens.
- Colon
  - In most cases, it is a sentence delimiter  $\Rightarrow$  separate tokens.
  - But it may be part of expressions (e.g. 12:30)  $\Rightarrow$  part of a token.
- And many, many other ambiguities...

---

### 3.1.2 Segmentation in other Languages

---

- In Chinese, there are no spaces.
- Beyond that, sentences may have completely different meanings when splitting on the wrong position. Chinese is context-sensitive.

---

### 3.1.3 Software

---

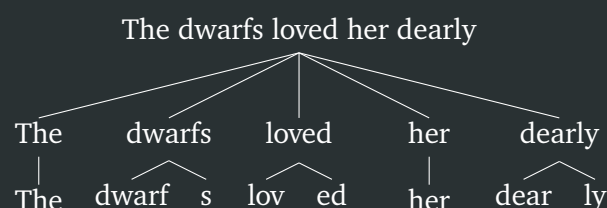
- Stanford CoreNLP and Word Segmenter (Java)
- OpenNLP Tokenizer (Java)
- LanguageTool Segmenter (Java)
- DKPro Core (Java)
- Natural Language Processing Toolkit (NLTK) (Python)

---

## 3.2 Morphology

---

- Morphology is about how single words are composed and is the study of word forms and word formation.
- Single words are composed of morphemes.
- *Morphemes* are the smallest unit a word can be split into.
- For instance:



---

### 3.2.1 Bases and Affixes

---

- Free morpheme
  - Morphemes that can appear in isolation.
  - Example: Both words “cat” and “cats” can appear in isolation.
- Bound morpheme, affix
  - Morphemes that can not appear in isolation, like the suffix “-s” in “cats”.
  - These are often used for inflection (plural, 3rd person singular, past, ...).
- Bases, stems
  - The minimal free morpheme of a word is called the *stem* of the word.
  - These stems carry the main meaning of the word.
  - Example: The stem of “cats” is “cat”.

---

### Types of Affixes

---

**Suffix** Appears after the base.

Example: “cat + s”

**Prefix** Appears before in the base.

Example: “un + true”

**Infix** Appears inside the base.

Example: “fan + bloody + tastic”

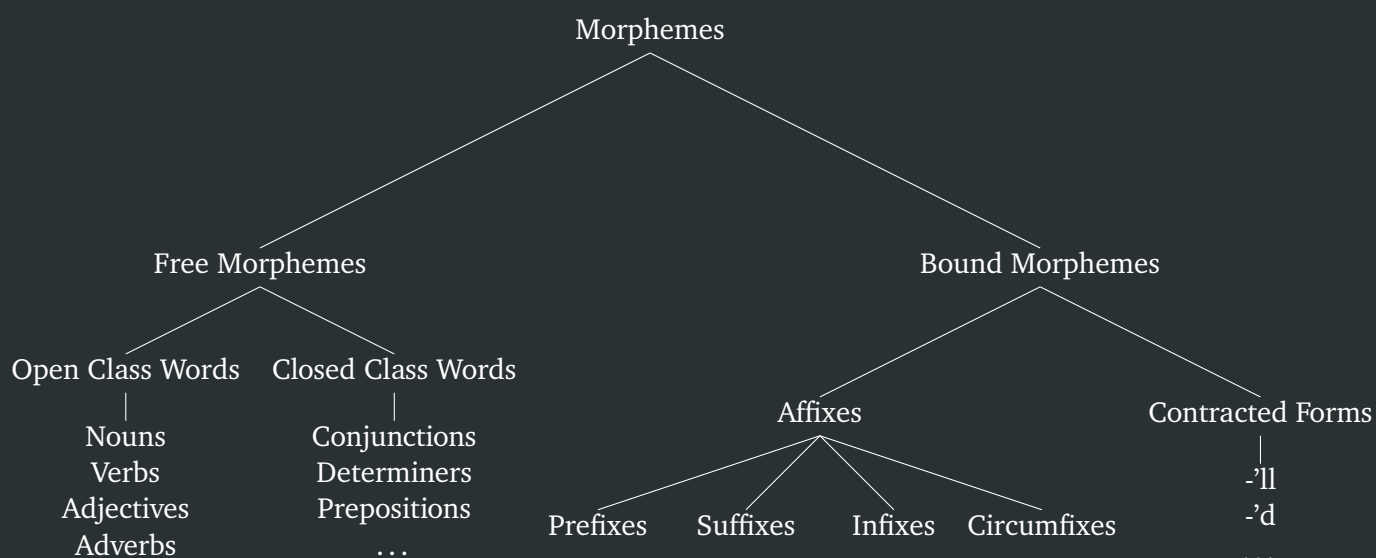
**Circumfix** Appears around (on both sides of) the base.

Example: “ge + sag + t”

---

### 3.2.2 Classification of Morphemes

---

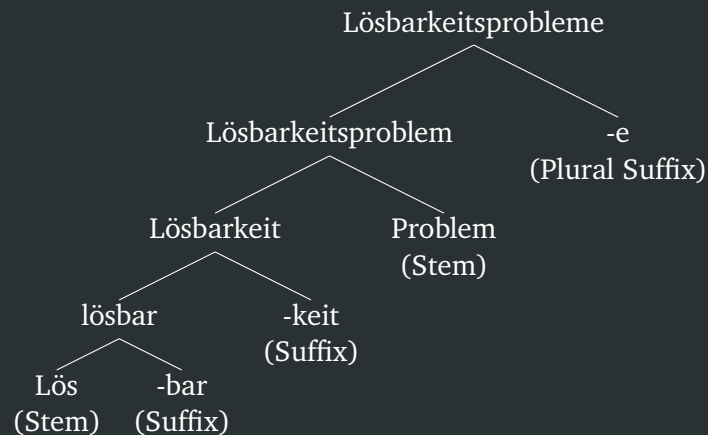




---

## Morphological Analysis

---



---

### 3.2.3 Word Formation

---

**Derivation** Forming new words from a free morpheme and an affix.

**Conversion** Also called *zero derivation*; Forming new words by changing the meaning of a word an affix, for example by modifying the stem.

Example: to talk → a talk; to host a party → the host; ...

**Composition** Also called *compounding*; Forming new words by combining multiple free morphemes and linking elements.

---

### Decompounding

---

- Splitting compounds into the creating parts.
- Example: “Bücherregale” → Bücher + Regale
- This may be complicated as splitting on different positions may lead to different meanings:
  - Kulturteilen → Kultur + teilen
  - Kulturteilen → Kult + urteilen

---

### 3.2.4 Morphology in other Languages

---

- In Turkish, only affixes are combined to form complete words.
- This is extremely hard to split.

---

### 3.2.5 Morphological Normalization

---

Normalizing a word is about to find a single canonical representation of a word. This gives multiple approaches:

- Stemming
- Lemmatization
- Morphological analysis

---

## Stemming

---

- Algorithmic approach to bring any word in its stemmed form.
- Words with the same morphological family should end up in similar stems.
- Stemming does not differentiate between stemming and derivation.
- The stemmed results may not be genuine word forms!
- Examples (Porter's Stemmer):

Original		Stemmed Word
visible	→	visibl
visibility	→	visibl
vision	→	vision
visionary	→	visionari
visioner	→	vision
visual	→	visual

## Errors

**Under-stemming** The stemmer does not transform related word to the same stem.

- adhere → adher
- adhesion → adhes

**Over-stemming** The stemmer can not distinguish between actual different words.

- appendicitis → append
- append → append

## Ambiguity: Homographs

- There are words that have different meanings, like:
  - Saw - The singular form of the noun "saw".
  - Saw - The past form of the present verb "see".
- It is not possible to handle these cases with stemming only, the word category (noun, verb, adverb, ...) also has to be identified.

---

## Lemmatization

---

- "Undo" the inflection changes and determine the actual base form.
- This typically requires information about the part of speech:
  - Verb: left → leave      saw → see
  - Noun: left → left      saw → saw
- Lemmatization has to deal with irregular forms:

- sing, sang, sung → sing
- indices → index
- Bäume → Baum

---

### Stemming, Lemmatization, Morphological Analysis

---

Original	Stemmed	Lemmatized	Analysis
visibilities	visibl	visibility	+PL
adhere	adher	adhere	
adhesion	adhes	adhesion	
appendicitis	append	appendicitis	+PL
oxen	oxen	ox	+PL
indices	indic	index	+PL
swum	swum	swim	+PP

---

### 3.2.6 Software

---

- Stanford CoreNLP (Java)
- SMOR Morphological Analysis for German
- Language Tool Lemmatizer (Java)
- DKPro Core (Java)
- Natural Language Processing Toolkit (NLTK) (Python)

---

## 3.3 Syntax

---

- The *Syntax* refers to the way multiple words are arranged together.
- There are infinite way to form word together to make up a sentence.
- Humans can understand most sentences they never heard before.

---

### 3.3.1 Part of Speech Tagging (POS Tagging)

---

- POS tagging is the process of assigning each word of a text its lexical class:

$\underbrace{\text{The}}$   
Determiner

 $\underbrace{\text{dwarfs}}$   
Noun

 $\underbrace{\text{loved}}$   
Verb

 $\underbrace{\text{her}}$   
Pronoun

 $\underbrace{\text{dearly}}$   
Adjective

---

### Parts of Speech

---

- The most commonly used word classes are:

**N** Noun

**V** Verb  
**ADJ** Adjective  
**ADV** Adverb  
**P** Preposition  
**PRO** Pronoun,  
**DET** Determiner

- These so-called *tagsets* differ for every language and may be really fine-grained or rough.
- Common tagset sizes:
  - English: 139
  - Czech: 970
  - Estonian: 476
  - Hungarian: 401
  - Romanian: 486
  - Slovene: 1033
- Data that is POS-tagged provides valuable information about:
  - the word formation → lemmatization, morphological analysis
  - a word and its possible neighbors → language models
  - the correct pronunciation → speech synthesis
  - its intended sense → word sense disambiguation
  - the meaning of a sentence → shallow parsing
- Example:

Word	Lemmatized	POS-Tag
the	the	+DET
girl	girl	+NOUN
kissed	kiss	+VPAST
the	the	+DET
boy	boy	+NOUN
on	on	+PREP
the	the	+DET
cheek	cheek	+NOUN

### Example Tagsets

- Penn Treebank Tagset: <http://www.clips.ua.ac.be/pages/mb-sp-tags>
- Stuttgart-Tübingen Tagset: <http://www.ims.uni-stuttgart.de/forschung/ressourcen/lexika/TagSets/stts-table.html>

---

## Ambiguity

- Often one word may have multiple possibilities for the POS-tag:
  - The **back** door is open. → ADJ
  - Peter has a scan on his **back**. → NOUN
  - She tried to win the voters **back**. → ADVERB
  - He promised to **back** the bill. → VERB

---

## Approaches to POS Tagging

- Rule-based Tagging
  - Create a dictionary word form → POS tags.
  - Then apply or automatically learn transformation rules to improve accuracy.
  - For yet unknown words, use the most frequent POS tag (e.g. NOUN).
- Probabilistic Tagging
  - Estimate the probability that a word has a specific tag → assign the tag with the highest probability.
  - Probabilities are learned from manually labeled data.

---

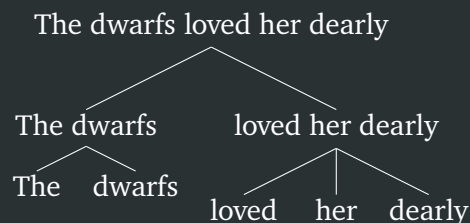
## Software

- **Stanford CoreNLP** (Java)
- **TreeTagger**
- **Mate Tools** (Java)
- **DKPro Core** (Java)
- **Natural Language Processing Toolkit (NLTK)** (Python)

---

### 3.3.2 Parsing

Parsing is about determining the grammatical structure of a sentence.



---

## Phrase Structure Grammars

---

- The *phrase structure grammar* is about representing a sentence by decomposing it into its constituents.
- A *constituent* is a group of words that behaves like a single unit. In phrase structure grammars, these are mostly phrases.  
Tests are used to identify constituents:
  - Example: The dog ate a **cookie**.
  - Substitution: The dog ate **it**.
  - Movement: **A cookie** was eaten by the dog.
  - Coordination: The dog ate a **cookie** and a sausage.
  - Question: What did the dog eat? **A cookie**.

---

## Phrase Types

---

**Noun Phrase (NP)** Noun as head.

Example: “the black **cat**”; “a **cat** on the mat”

**Prepositional Phrase (PP)** Preposition as head.

Example: “**in** love”; “**over** the rainbow”

**Verb Phrase (VP)** Verb as head.

Example: “**eat** cheese”; “**jump** up and down”

**Adjectival Phrase (AP)** Adjective as head.

Example: “**full** of toys”; “**fraught** with guilt”

**Adverbial Phrase (AdbP)** Adverb as head.

Example: “**dearly**”; “very **carefully**”

- The *phrase head* determines the syntactical type of the sentence.
- A *grammatical modifier* is an optional element of a phrase.
  - “Modifies” the meaning (e.g. “the ball” vs. “the red ball”).
  - A *premodifier* appears in front of the head.
  - A *postmodifier* appears after the head.
  - These are often adjectives, adverbs, prepositional phrases, ...

---

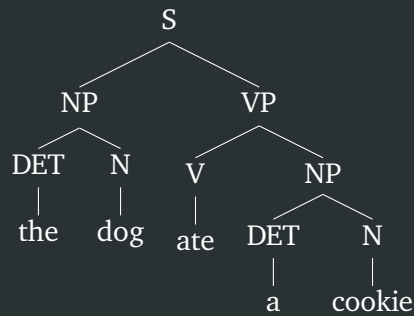
## Phrase Parsing

---

- For a given sentence, check if it can be generated by a given grammar.
- Sentences that can be generated by a grammar are called *grammatical sentences*.
- *Ungrammatical sentences* are unacceptable/uninterpretable with respect to the grammar.

- Even more important than that can it be generated by a grammar is, how can it be generated? That is, how can a sentence be decomposed into phrases.
- This step produces a *parse tree*.

**Parse Tree** The following is the parse tree for the sentence “the dog ate a cookie”.



### Alternative Notations

- Bracket notation:

$[S [NP [DET the] [N dog]] [VP [V ate] [NP [DET a] [N cookie]]]]$

- Parenthesized notation:

```

(S
  (NP
    (DET the)
    (N dog) )
  (VP
    (V ate)
    (NP
      (DET a)
      (N cookie) ) ) )
  
```

---

### Context-Free Grammars (CFG)

---

Context-free grammar  $G = (T, N, S, R)$  with:

- Terminals  $T$ ,
- Non-terminals  $N$ ,
- Start symbol  $S$  and
- production rules  $R$  with elements in the form  $X \rightarrow \gamma$ , where  $X \in N$  and  $\gamma \in (T \cup N)^*$ .

The grammar  $G$  generates language  $L(G)$ .

---

### Syntactical Ambiguity

---

Sometimes a sentence has more than one valid syntactical structure.

- **Attachment Ambiguity:** A constituent can be added to the parse tree at different locations.  
Example:

- “I shot an elephant in my pants.”

$\xRightarrow{A?} VP \rightarrow VP NP \Rightarrow \underbrace{I \text{ shot}}_{VP} \underbrace{an \text{ elephant in my pants.}}_{NP}$

$\xRightarrow{B?} VP \rightarrow NP NP \Rightarrow \underbrace{I \text{ shot an elephant}}_{NP} \underbrace{in my pants.}_{NP}$

- **Coordination Ambiguity:** Varying scope of the conjunction.  
Example:

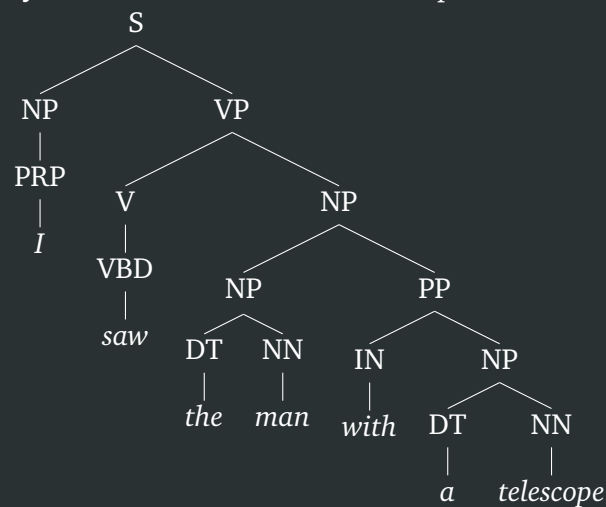
- “Black cats and dogs like to play.”

$\xRightarrow{A?} NP \rightarrow AP VP \underbrace{Black \text{ cats and dogs}}_{AP} \underbrace{like \text{ to play.}}_{VP}$

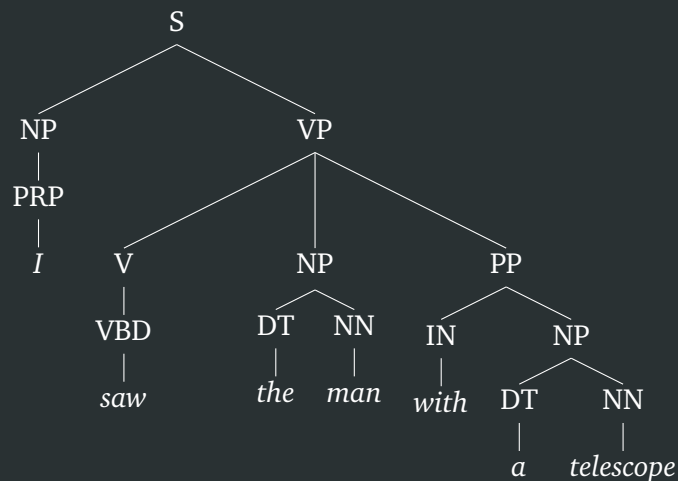
$\xRightarrow{B?} NP \rightarrow AP NP VP \underbrace{Black \text{ cats and dogs}}_{AP} \underbrace{like \text{ to play.}}_{VP}$

- **Garden path sentence:** A grammatical sentence where it is not possible to apply obvious grammatical rules.  
– “The old man the boat.”

**Example: Attachment Ambiguity** “I saw the man with a telescope.”







## Dependency Grammar

- A *dependency grammar* is a different kind of syntactic representation.
- Not based on phrases but on binary relations between words:

*grammatical-function*(**head**, dependent)

- Examples:
  - Subject: “Sue **watched** the man at the next table.”
  - Direct Object: “Sue **watched** the man at the next table.”
  - Determiner: “Sue watched the man at the next table.”
  - ...
- Abstract from word order.

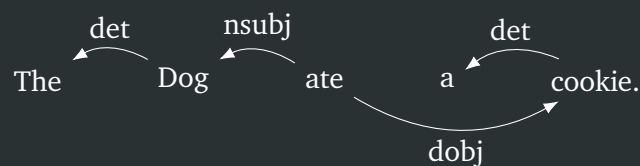
**Example** “The dog ate a cookie.”

- Textual notation:

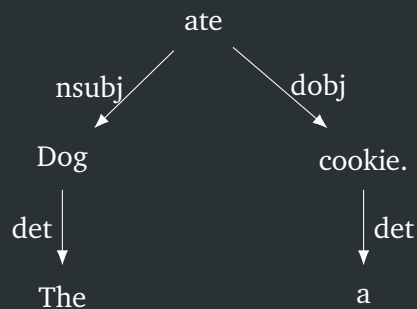
```

det(dog, The)
nsubj(ate, dog)
root(ROOT, ate)
det(cookie, a)
dobj(ate, cookie)
  
```

- Line notation:



- Tree notation:



---

## Software

---

- **Stanford Parser** (Java, PHP, Python, Ruby)
- **Berkeley Parser** (Java)
- **Mate Tools** (Java)
- **DKPro Core** (Java)
- **Natural Language Processing Toolkit (NLTK)** (Python)

---

## 3.4 Semantic

---

- *Semantics* is the study of meaning.
- *Lexical Semantics* is the study of the meaning of lexical terms (e.g. word).
- *Structure Semantics* is the study of the relationships between the meaning of lexical items in a larger context (e.g. phrases, sentences, documents).

---

### 3.4.1 Ambiguity

---

- Lexical Ambiguity: “He hit the ball with a **bat**.”
  1. **Bat** could refer to the animal or
  2. **Bat** could refer to a baseball racket.
- Syntactical Ambiguity: “If you love money problems show up.”
  1. **If you love**, money problems show up.
  2. **If you love money**, problems show up.
  3. **If you love money problems**, show up.
- Syntactic and lexical ambiguity: “Time flies like an arrow.”
  1. Time moves quickly.
  2. Measure the speed of flies the same way as the speed of an arrow.

- 
3. Measure the speed of flies like an arrow would.
  4. Measure the speed of flies that behave/look/... like an arrow.
  5. The insect type “time flies” enjoy a single arrow (like “fruit flies like a banana”).
  6. Each of the “time flies” individually enjoys a different arrow.
  7. A concrete object (e.g. Time magazine) travels through the air in an arrow-like manner.
  8. ...

---

### 3.4.2 Word Sense Disambiguation

---

- *Word sense disambiguation* is about to remove the the ambiguity by putting the single words into the context of the sentence/text/... and using background information.

---

## 4 Text Corpora, Lexical Resources and Knowledge Bases

---

### 4.1 Text Corpora

---

- A *Text Corpus* is collection of text used to train or evaluate a system.
- Examples:
  - Brown Corpus: 500 texts, 15 genres,  $\approx$  1 mio. words
  - Wall Street Journal Campus: news,  $\approx$  30 mio. words
  - British National Corpus: balanced among genres,  $\approx$  100 mio. words
  - WaCKy Corpora: large web corpora in multiple language
- A *Parallel Corpora* is a corpora with their documents translated into two or more languages.
  - Ideally it is sentence-aligned.
  - Used for machine translation or cross-lingual information retrieval.
  - Examples:
    - \* Canadian Hansard
    - \* EuroParl
- A (good) corpus provides information about:
  - real usage examples,
  - a view on what is common and typical (e.g. count frequencies),
  - a comprehensive and balanced view on language (if the corpus was constructed correspondingly),
  - recent changes in a language (if the corpus is continuously updated) and
  - a basis for reproducible experimental results.

---

#### 4.1.1 Parameters of a Corpus

---

- Language (Monolingual, Multilingual, Parallel)
- Communication (Written, Spoken, Mix)
- Size
- Annotations
- Static  $\leftrightarrow$  Dynamic

- 
- Genre/Text Type (news, novels, social media text, personal conversations, ...)
  - Domain/Topic (education, biology, information science, ...)
  - Time of Compilation/Creation Time

---

## 4.1.2 Content of a Corpus

---

### Corpus Annotation

---

- *Corpus Annotations* are enrichments of a corpus with various types of information.
- Annotations can be done on different levels, e.g.:
  - Word: part of speech, lemma, sense  
These require segmentation of the document texts into parts.
  - Phrase: named entities, multi-word expressions
  - Sentence: sentence boundaries, syntactic tree
  - Discourse: co-referential chains, discourse segments

---

### Frequencies

---

- **Collocations:** Sequence of words that occur together usually often.
- **Distributional Hypothesis:** Word in the same context might have similar meanings.

---

## 4.1.3 Distributional Hypothesis

---

- Main idea:

“You shall know a word by the company it keeps” (Firth, 1957: p. 11)
- Words that occur in the same contexts have similar meanings
- She **adores green** paint.
  - verbs expressing admiration: adores, loves, likes, ...
  - colors: green, blue, red, ...
- Word sense disambiguation
- Word space models: Compute word or text similarity
  - “cab – van” is more similar than “cab – database”
- It is important for:
  - Document classification/clustering
  - Information retrieval
  - Question answering (find similar questions)
  - Paraphrasing assistance
  - Word embeddings and deep learning

---

#### 4.1.4 Corpus Creation

---

1. Retrieve and store original documents.
2. Convert these document to plain text.
3. Segment the documents into chapters, sentences, phrases, words, ...
4. Create manual and/or automatic annotations.
5. Store all data in a reusable format.
6. Analyze and use the corpus.

Important: Store all intermediate results and document the steps for reproducible results.

---

#### Storing Annotations

---

There are two main ways to store annotations:

1. Inline Annotations
  - The annotations are directly added to the text.
  - This changes the format of the original.
2. Stand-off Annotations
  - The annotations are stored in a separate file.
  - This leaves the original file as is, but the annotation data has to be restored.

---

#### 4.1.5 Corpus Query Languages

---

- The standard operations in SQL are exact and substring matches.
- But with complex corpus queries, it should be possible to find things like:
  - All words with 6 letters starting with A or K that end with a vowel.
  - Sentences containing two words at any position.
  - ...
- Possible solutions:
  - SQL functions
  - Pattern matching and regular expressions
  - Corpus Query Processor (CQP) language

---

#### 4.1.6 Example Corpora

---

##### Treebanks: Penn Treebank

- Mostly based on Wall Street Journal text.
- Annotations: POS, syntactic parse tree

---

### **Brown Corpus: POS-Tagged Text**

- Inline POS-tags.

### **Corpora in Relational Databases**

- Example: Leibniz Corpus Collection

### **XML Corpora: British National Corpus**

- Stored in XML.

### **Getting Corpora**

- Major organizations:
  - Linguistic Data Consortium
  - European Language Resources Association
- Freely accessible corpora:
  - Oxford Text Archive
  - Electronic Text Archive
  - Project Gutenberg
  - Leibniz Corpora Collection
  - NLTK Corpora
  - Institut für Deutsche Sprache

---

## **4.2 Lexical Resources and Knowledge Bases**

---

Types of lexical resources:

- Dictionaries
- Encyclopedias
- Thesauri
- Wordnets
- ...

---

### 4.2.1 Lexical Resources ↔ Corpora

---

#### Corpus

Collected from real-world text/speech.  
Contains multiple occurrences of a lemma.  
Frequent phenomena occur more often.  
Show how language is used  
Provides typical context and frequencies.

#### Lexical Resource

Derived from corpora (aggregated view).  
A lemma usually only occurs once.  
Rare and frequent phenomena are treated equally.  
Describes how language is used.  
Provide meta information (e.g. sense definition)

---

### 4.2.2 Meaning, Lexical Ambiguity, Synonymy

---

- A single lexical entry may have multiple meanings.
- The meanings described in a dictionary are called *word senses*.
- Example:
  - The word “bat” could either refer
  - to a racket or
  - to an animal.
- Vice versa, the same sense may have multiple associated words (synonyms).

---

### 4.2.3 Relation Types

---

Relation Type	Description	Relations	Example
synonymy	same meaning	antonym of antonymy	stack is synonym of pile
antonymy	opposite meaning	antonym of synonymy	rich is antonym of poor
hypernymy	broader meaning	antonym of hyponymy	vehicle is hypernym of car
hyponymy	narrower meaning	antonym of hypernym	taxi is hyponym of car
co-hypernymy	same hypernym	hypernym, synonym	cat and dog are co-hyponyms of pet
troponymy	“hyponymy for verbs”	related to hypernymy	(to) nap is troponym of (to) sleep
holonymy	X is the whole of Y	antonym of meronymy	car is holonym of door
meronymy	X is a part of Y	antonym of holonymy	door is meronym of car
seeAlso	related meaning		bread and baker are related

---

### 4.2.4 Wordnets

---

- Princeton WordNet
- GermaNet
- EuroWordNet



---

# 5 Information Retrieval

---

---

## 5.1 Overview

---

- *Information Retrieval* is about retrieving information from unstructured data best matching a given query.
- Exemplary scenarios:
  - Web search
  - E-Mail search
  - Search in knowledge base or wiki
  - ...

---

### 5.1.1 Basic Concepts

---

- *Information Need*, also called *intent*, is the state of a person requiring information for solving a problem.
- The system-interpretable information need is formulated as a *Query*, e.g. the keywords entered into a search engine.
- The *Relevance* is about how relevant a particular document is to a particular query. The task of IR is to retrieve the most relevant documents.

---

### 5.1.2 IR Engine vs. DBMS

---

- The DBMS relies on structure in the data (tables, columns, ...).
- An IR engine works with unstructured data like text.
- Thus, an IR engine supports more complex finding operations like “documents in which word1 is next to word2”.
- LIKE-Queries in DBMS require linear searches on the text and full table scans  $\implies$  inefficient.

---

### 5.1.3 Core Challenges in IR

---

- **Number of result:** There are millions of documents in the web, what are the relevant ones?
- **Relevance:** Some results may be better than others, how to separate them?
- **Intent:** The user may not use an ideal query for their information need, how to find what the user wants rather than what the user says he wants?
- **Lexical gap:** Searching for “get rid of mice” should lead to documents about mouse traps.
- **Ambiguity:** Mouse may be an animal or an input device, how to separate?

---

#### 5.1.4 IR Activities

---

- *Indexing*: Create a simple document representation to enhance search.
- *Searching*: Interpret the query and return matching documents, sorted by relevance, ranked by results, improved by user feedback, ....

---

### 5.2 Boolean Retrieval

---

- *Boolean Retrieval* is the most simple retrieval model.
- One document is considered to be a *set of words* with no duplicates.
- Searching for documents be like:
  - contains a word,
  - does not contain a word,
  - contains word1 and word2,
  - contains word1 or word2

---

#### 5.2.1 Term-Document Matrix

---

Whether a word is contained in a document can be represented as a 1, not containing as a 0. This leads to the following matrix (as an example), the so-called *Term-Document Matrix*:

Term $t$	Webshop deadfall traps	Wikipedia: mouse- traps	Webshop live- capture traps 1	Wikipedia: traps	tips to get rid of rodents	Webshop live- capture traps 2	bear hunting overview
mouse	1	1	1	0	0	0	0
trap	1	1	1	1	0	1	0
not	1	0	1	0	0	0	0
hurt	0	1	1	0	1	0	0
dead	1	0	0	1	0	0	0
alive	0	0	0	1	1	1	0
bear	0	0	0	0	0	0	1
rodent	0	1	0	0	1	1	0

- One column represents one document and one row represents one term.
- As said above, a 1 means the term is in the document, a 0 means the term is not.
- When searching for a term, say  $(\text{mouse} \vee \text{rodent}) \wedge \neg \text{hurt}$ , the searching can be processed using bit-wise

logic:

$$\begin{aligned} & (\text{mouse} \vee \text{rodent}) \wedge \neg \text{hurt} \\ \simeq & (1110000 \vee 0100110) \wedge \neg 0110100 \\ = & 1110110 \wedge 1001011 \\ = & 1000010 \\ \simeq & \{\text{Webshop deadfall traps}, \text{Webshop live-capture traps 2}\} \end{aligned}$$

- The result bit pattern, 1000010, refers to the documents to select: “Webshop deadfall traps” and “Webshop live-capture traps 2”.

---

## Scaling, Inverted Matrix

---

- Saving the matrix as described above would lead to high memory usages to save lots of 0s.
- There must be a better way to save the data.
- Idea: Just save the 1s.
- This can be done using an *Inverted Index*:
  - Each document is assigned a number (say from d1 to d7 for the example above).
  - Then, each term is assigned a set of document numbers the term appears in.
  - Using this technique, one the appearances of a word are saved rather than saving a bunch of “does not appear in”, “does not appear in”, “does not appear in”, ....

	Webshop deadfall traps	Wikipedia: mouse-traps	Webshop live-capture traps 1	Wikipedia: traps	tips to get rid of rodents	Webshop live-capture traps 2	bear hunting overview
	d1	d2	d3	d4	d5	d6	d7

mouse	d1	d2	d3				
trap	d1	d2	d3	d4	d6		
not	d1	d3					
hurt	d2	d3	d5				
dead	d1	d4					
alive	d4	d5	d6				
bear	d1						
rodent	d2	d5	d6				

---

## 5.3 Vector Space Model

---

---

### 5.3.1 Boolean Search Issues

---

#### 1. Syntax Needed

- The user has to write Boolean expressions.
- This is nearly impossible for the average user and time-consuming for the expert user.
- **Solution:** use keyword-bases search instead (e.g. mouse trap  $\simeq$  mouse  $\wedge$  trap).

## 2. “Feast or Famine”

- Boolean queries often give either too few ( $\approx 0$ ) or too many ( $\gg 1000$ ) results.
- Neither of these is useful.
- **Solution:** Ranked retrieval.

### 5.3.2 Ranked Retrieval

- In the Boolean retrieval, a term was either in a document or not.
- But a document containing one word multiple times may be more relevant than a document containing the term a single time.
- Idea: Sort (rank) the results by relevance.
- Often, the best 10 results are the most relevant. Others can be discarded.
- This solves the “feast or famine”-problem.

### Time-Document Matrix Weights

- Instead of weighing the matrix components as 1 or 0, each field can be assigned the *Term Frequency*, the occurrence count of a word.
- That way, the matrix is weighted and can distinguish between not-so-relevant and relevant results.

Term $t$	Webshop deadfall traps	Wikipedia: mouse- traps	Webshop live- capture traps 1	Wikipedia: traps	tips to get rid of rodents	Webshop live- capture traps 2	bear hunting overview
mouse	12	53	3	0	0	0	0
trap	5	11	34	10	0	8	0
not	231	0	53	0	0	0	0
hurt	0	1	22	0	2	0	0
dead	50	0	0	3	0	0	0
alive	0	0	0	5	7	3	0
bear	0	0	0	0	0	0	32
rodent	0	45	0	0	3	5	0

---

### 5.3.3 Set of Words, Bag of Words

---

- Set of Words
  - Disregards word order, grammar, meaning, ...
  - A document either contains a term or not.
  - Each term is assumed to occur only once or never.
- Bag of Words
  - Disregards word order, grammar, meaning, ...
  - A token may occur multiple times in a document.
  - The term frequency (TF) function represents the occurrence counts.

---

### 5.3.4 Vector Space Model (VSM)

---

- Each document in the TD-matrix can be represented as an  $n$ -dimensional vector, where  $n$  is the number of terms.
- This puts the TD-matrix into a vector space which then allows to apply linear algebra operations.

**Example** *In this example, the dimension of the VCM is reduced to 2 to ease the visualization, a 8-dimensional vector space is kinda complicated to visualize.*

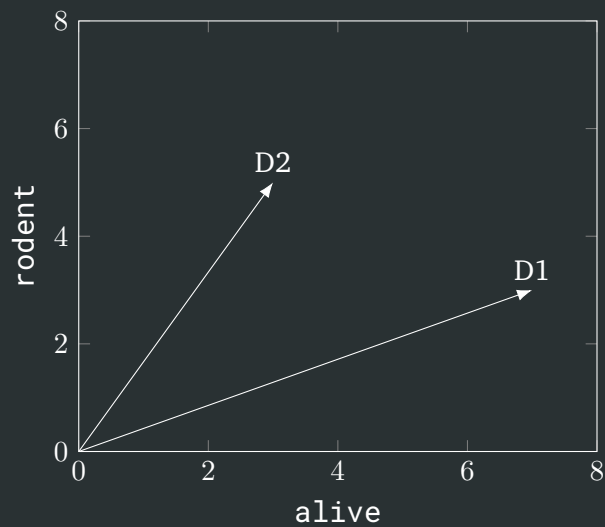
Given the following TD-matrix (reduced to 2 dimensions as explained):

Term $t$	tips to get rid of rodents (D1)	Webshop live- capture traps 2 (D2)
alive	7	3
rodent	3	5

So, the documents D1 and D2 can be represented using vectors:

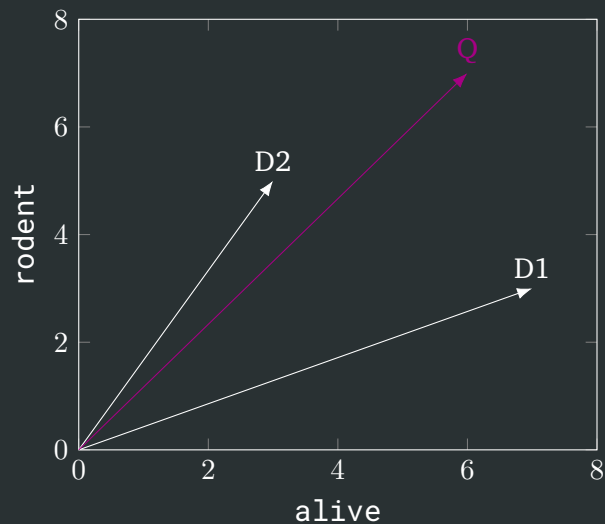
$$D1 = \begin{bmatrix} 7 \\ 3 \end{bmatrix}, \quad D2 = \begin{bmatrix} 3 \\ 5 \end{bmatrix}$$

In a two-dimensional vector space, this gives the following graph:



### 5.3.5 Search the VSM

Given the above vector space, it is possible to trace out a query  $Q$  in the vector space:



### Vector Similarity

- To find results in the VSM, the similarity between the vectors has to be calculated.
- The first approach might be to use the **Euclidean Similarity**, which is to calculate the distance between the query vector and every other vector:

$$\text{Similarity}(D, Q) = |Q - D|$$

Problem: The distance differs depending on how often a keyword is entered into the query.

- The much better approach is not measure the angle between the query vector and every other vector:

$$\text{Similarity}(D, Q) = \frac{\langle Q | D \rangle}{|Q| \cdot |D|}$$

- Example:

- $q = (0; 5; 0; 1; 0)$

- $v = (0; 2; 3; 2; 1)$

$$\text{sim}(q, v) = \frac{0 \cdot 0 + 5 \cdot 2 + 0 \cdot 3 + 1 \cdot 2 + 0 \cdot 1}{\sqrt{0^2 + 5^2 + 0^2 + 1^2 + 0^2} \cdot \sqrt{0^2 + 2^2 + 3^2 + 2^2 + 1^2}} = \frac{12}{\sqrt{26} \cdot \sqrt{18}} \approx 0.55$$

---

### 5.3.6 Vector Weights

---

- Let  $v = (w_{t_1}, \dots, w_{t_n})$  be the vector for the document  $d$  with terms  $t_1$  to  $t_n$ .
- Let  $\text{tf}_d(t)$  be the occurrence count of  $t$  in  $d$ , the *Term Frequency*.

#### Binary

$$w_t = \begin{cases} 1 & t \in d \\ 0 & \text{otherwise} \end{cases}$$

- Discussion: Isn't there a relevance difference between documents with one occurrence and a document with thousand occurrences?

#### Term Frequencies

$$w_t = \begin{cases} \text{tf}_d(t) & t \in d \\ 0 & \text{otherwise} \end{cases}$$

- Discussion: Is a document containing the search term 1000 times really 100 times more relevant than a document containing the search term 10 times?

#### Normalized Term Frequencies

$$w_t = \begin{cases} \log_{10}(\text{tf}_d(t)) + 1 & t \in d \\ 0 & \text{otherwise} \end{cases}$$

- Discussion: Determiners occur very often, but are not really relevant. Words occurring rarely over a set of document would be useful.
- Example: the, a and am appear very often and are probably not as important as paint, colour, mouse, ...

**Document Frequency** Let  $\text{df}(t)$  be the count of documents containing the term  $t$ , the *Document Frequency*.

$$w_t = \begin{cases} f(\log_{10}(\text{tf}_d(t)) + 1, \text{df}(t)) & t \in d \\ 0 & \text{otherwise} \end{cases}$$

with some function  $f(\cdot, \cdot)$ .

- Discussion; TF should be maximized, DF should be minimized  $\implies$  hard to perform on a single output number.

---

## Inverse Document Frequency

---

Let  $\text{idf}(t) := \log_{10} \left( \frac{|D|}{\text{df}(t)} \right)$  be the *Inverse Document Frequency* with the document set  $D$ .

$$w_t = \begin{cases} f(\log_{10}(\text{tf}_d(t)) + 1, \text{idf}(t)) & t \in d \\ 0 & \text{otherwise} \end{cases}$$

with some function  $f(\cdot, \cdot)$ .

---

## Concrete Vector Weights: TF.IDF

---

$$w_t = \begin{cases} (\log_{10}(\text{tf}_d(t)) + 1) \cdot \text{idf}(t) & t \in d \\ 0 & \text{otherwise} \end{cases}$$

Of course, other normalization strategies are also possible, like:

$$w_t = \begin{cases} \text{tf}_d(t) \cdot \text{idf}(t) & t \in d \\ 0 & \text{otherwise} \end{cases}$$

---

### 5.3.7 VSM Retrieval: Example

---

Term $t$	Webshop deadfall traps	Wikipedia: mouse- traps	Webshop live- capture traps 1	Wiki- pedia: traps	tips to get rid of rodents	Webshop live- capture traps 2	bear hunting overview	Query Q
mouse	12	53	3	0	0	0	0	1
trap	5	11	34	10	0	8	0	1
not	231	0	53	0	0	0	0	1
hurt	0	1	22	0	2	0	0	1
dead	50	0	0	3	0	0	0	0
alive	0	0	0	5	7	3	0	0
bear	0	0	0	0	0	0	32	0
rodent	0	45	0	0	3	5	0	0
$ d $	237	70	67	12	8	9	6	2
$\text{sim}(D, Q)$	0.52	0.46	0.84	0.42	0.13	0.44	0	

---

## 5.4 Information Retrieval Evaluation

---

- The evaluation of a retrieval system is extremely important to check how good the retrieval system really is.
- The basic measurements are:
  - *Precision* - The fraction of the retrieved documents that are relevant to the user's information need.

$$P = \frac{|\text{Retrieved} \cap \text{Relevant}|}{|\text{Retrieved}|} = \frac{|\text{True Positives}|}{|\text{True Positives}| + |\text{False Positives}|}$$



- *Recall* - The fraction of the relevant documents in the collection that are retrieved.

$$R = \frac{|\text{Retrieved} \cap \text{Relevant}|}{|\text{Relevant}|} = \frac{|\text{True Positives}|}{|\text{True Positives}| + |\text{False Negatives}|}$$

- Confusion Matrix

	Relevant	Irrelevant	$\Sigma$
Retrieved	TP = 2	FP = 1	3
Not Retrieved	FN = 3	TN = ??	
$\Sigma$	5		

---

#### 5.4.1 $F_1$ Score

---

The  $F_1$ -score is the weighted average of precision and recall:

$$F_1 = \frac{2PR}{P + R} \quad (\text{Harmonic Mean of } P \text{ and } R)$$

**Generalization:  $F_\beta$  Score**

$$F_\beta = \frac{(1 + \beta^2)PR}{\beta^2P + R}$$

---

#### 5.4.2 Cutoff and Precision at Rank

---

$$P@R = \frac{\text{\#Relevant Documents before } R}{R} \quad (\text{Precision at Rank})$$

---

## 6 Information Extraction and Classification

---

### 6.1 Information Extraction (IE)

---

**Given** A document collection.

**Goal** Extract structured knowledge, entities, facts, ....

---

#### 6.1.1 Information Retrieval vs. Information Extraction

---

- *Information Retrieval* finds documents that are relevant to a query.
  - Output: List of relevant documents.
  - A lot less difficult than IE, as “only” documents have to be found.
  - Domain-independent.
  - Query types are usually unconstrained (can contain any keyword).
  - Faster.
  - Less effective as the user wants answers, not documents.
- *Information Extraction* extracts structured information from an unstructured document.
  - Output: Set of structured facts extracted from documents.
  - Much more difficult than IR (has to do IR first and then extract facts).
  - Often domain-dependent.
  - Pre-defined query types (e.g. SQL).
  - Slower.
  - More effective as the user wants answers and gets answers.

---

### 6.2 Entity Recognition

---

---

#### 6.2.1 Entity Types

---

Commonly used entity types:

Type	Examples
ORGANIZATION	TU Darmstadt, TU Dresden
PERSON	Angela Merkel, Goethe
LOCATION	Darmstadt, Rhein, Luisenplatz
DATE	10.09.2018
TIME	10:00
MONEY	250 Euros
PERCENT	43%
FACILITY	London Bridge, Karo5
...	...

The definition of the concrete used entity types highly depends on the domain and on the task.

---

### 6.2.2 Challenges

---

- Entity vs. Non-Entity  
Example: Mobile Phone vs. Mobile, Alabama
- Coverage Issues  
Example: It is impossible to create a list of all persons, locations, ....
- Variation  
Example: John Smith vs. Mr. Smith, John vs. Mr. J. Smith
- Ambiguity  
Example: Darmstadt (German vs. US city)
- Time Dependency  
Example: The president of the TU Darmstadt (Prömel, Wörner, ...).
- Multi-word Expressions; The boundaries are often not clear.  
Example: Carlo und Karin Giersch-Stiftung an der TU Darmstadt
- Metonymy
  - A figure of speech in which an entity is not called by its name but by the name of any property intimately associated with it.
  - Examples:
    - \* Darmstadt won by a penalty goal in the last minute of the game.  
(Darmstadt does not refer to the city, but to the SV Darmstadt 98.)
    - \* She only reads Goethe and Schiller.  
(Refers to books of Goethe and Schiller, not their personalities.)

---

### 6.2.3 Approaches

---

#### List Lookup

---

- Recognize only entities that are stored in a list.
- For locations, these lists are called *gazetteers*.

- **Advantages**
  - Simple
  - Fast
  - Easy to adapt to a different domain
- **Disadvantages**
  - Lists have to be collected and maintained
  - Cannot deal with name variants and abbreviations
  - Cannot resolve ambiguity

---

### Rule-based Methods

---

- Most often, the rules are regular expressions.
- But: Extremely Labor-intensive.
  - It is easy to get a fairly good performance (e.g. 50% correct).
  - But it is incredible hard to get a good performance (e.g. > 70% correct).
  - It is very expensive to maintain and to adapt to new domains, tasks and languages.

---

### 6.2.4 Next Steps After Information Extraction

---

- After we extracted all these facts, what shall we do with this information?
- We save it in a knowledge base

---

### Knowledge Bases

---

- A knowledge base is a special kind of database for knowledge management. A knowledge base provides a means for information to be collected, organized, shared, searched and utilized.
- Collection of facts, which can be stored as SPO Triples (Subject, Predicate, Object)
- Example: “Leonard Nimoy was an actor who played the character Spock in the science-fiction movie Star Trek”

Subject	Predicate	Object
(LeonardNimoy,	profession,	Actor)
(LeonardNimoy,	starredIn,	StarTrek)
(LeonardNimoy,	played,	Spock)
(Spock	characterIn,	StarTrek)
(StarTrek	genre,	ScienceFiction)

- SPO Triples can be combined to form a graph
  - Nodes** Entities (all subjects and objects)
  - Directed edges** predicates
- This graph is called a **Knowledge Graph (KG)**

---

## Knowledge Base Construction Methods

---

- **Curated approaches**
  - Triples are created manually by a closed group of experts
  - (+) High quality
  - (-) Low scalability
- **Collaborative approaches**
  - Triples are created manually by an open group of volunteers
  - Examples: Freebase, Wikipedia, Wikidata
  - (+) Better scalability
  - (-) Still limited
- **Automated semi-structured approaches**
  - Triples are extracted automatically from semi-structured text
    - \* Infoboxes in Wikipedia
    - \* hand-crafted rules, learned rules or regular expressions
  - Examples: YAGO, DBPedia
  - (+) Large knowledge graphs
  - (-) Still cover only a small fraction of the Web
- **Automated unstructured approaches**
  - Triples are extracted automatically from unstructured text
  - Machine learning and natural language processing (NLP) techniques
  - Examples: NELL, Knowledge Vault
  - (+) huge knowledge graphs: “read the web”

---

## 6.3 Supervised vs. Probabilistic Machine Learning

---

- Supervised Machine Learning
  - Core Idea: Create a dataset and let the machine learn rules or statistical models to label unseen data.
- Corpus-based Probabilistic Methods
  - Core Idea: Estimate the probability for certain language structures based in large corpora.

---

### 6.3.1 Classification

---

- **Classification** is the task of choosing the correct **class label** for a given input (=instance) based in its features
- Examples:

- 
- Email spam classification
  - Categorizing news articles by topic
  - **Classification variants**
    - binary (2 classes) vs multi-class classification ( $>2$  classes)
    - multi-class problem can be decomposed using binary classifiers
    - single-label vs. multi-label classification, each instance may be assigned one vs. multiple class labels
    - sequence classification, a sequence of instances are jointly classified

---

### 6.3.2 General (Supervised) ML Workflow

---

1. Obtain training, development, test data
2. Represent the input (e.g., using features)
3. Select an algorithm and train the model
4. Evaluate the results; perform the task

---

#### Split the Data

---

- If possible: use separate datasets for development, testing and evaluation
- If not: divide one large dataset in three parts

**Training Set** Trains the model

**Test Set** Analyze errors, select features and optimize parameters on it

**Evaluation Set** Test on held-out data and evaluate how well the algorithm works, dont optimize on it