Anwendungen der KI

– Sommersemester 2018 –

Kapitel 04: Natural Language Processing

Prof. Dr. Adrian Ulges
B.Sc. Informatik (AI, ITS, MI, WI)
Fachbereich DCSM
Hochschule RheinMain

Outline

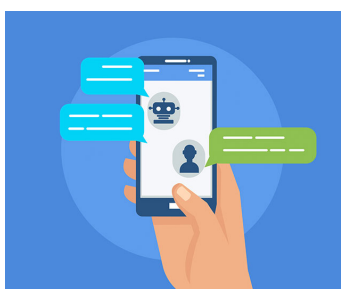
Natural Language Processing: Einordnung

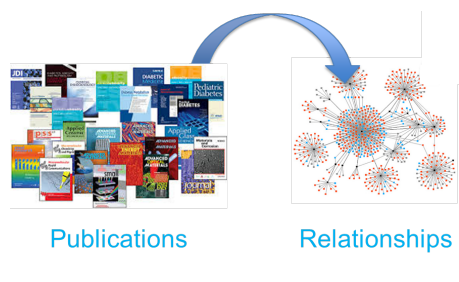
Computerlinguistik

- Ziel ist **erkenntniswissenschaftlich**: Besseres Verständnis für Sprache *(wie/wann hat sich Sprache X aus Sprache Y entwickelt? Wie dynamisch ist Sprache? Können wir Sprache synthetisieren? ...)*
- Der Computer als Werkzeug für Erkenntnisgewinn, z.B. um **statistische Modelle** auf größeren Korpora zu erzeugen.

Natural Language Processing (NLP

- **Automatisierte Verarbeitung** (gesprochener oder geschriebener) natürlicher Sprache
- Ziel ist **ingenieurswissenschaftlich**: Lösen praktischer Probleme *Suche+Empfehlung, Chatbots, Question Answering, Informationsextraktion, Übersetzung ...*
- **Fragestellungen**: Parsing, POS-Tagging, Erkennung benamter Objekte ("named entities") + Relationen, Themenerkennung, Sentimentanalyse ...





Natural Language Processing: Grundbegriffe

Morphologie

- Analyse der **bedeutungsgebenden Komponenten** eines Wortes (`walk - ed`)
- Beinhaltet z.B. *Stemming (siehe letzte Vorlesung)*.

Syntax *(griech. "zusammensetzen", "ordnen")*

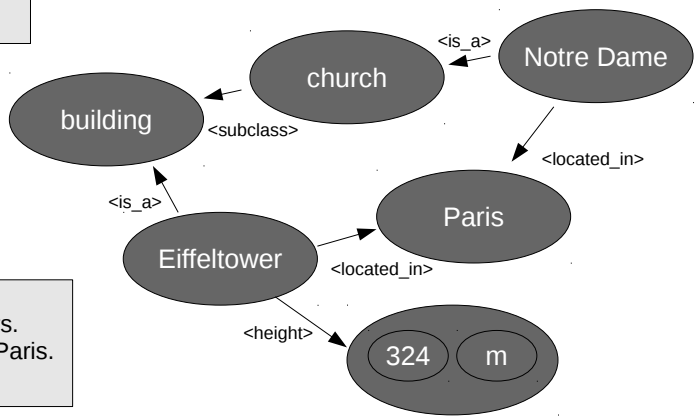
- Wie sind Worte angeordnet?
  Was ist die Struktur eines Satzes?

Semantik

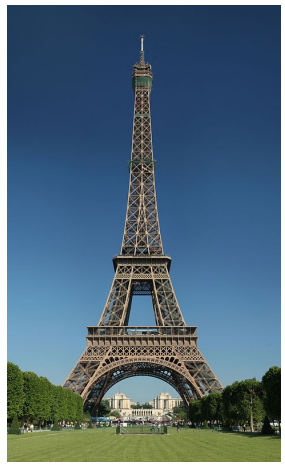
- Was sind die **Bedeutungen** eines Wortes / Satzes?
- **Ziel** vieler NLP-Techniken
  - **Klassifikation** von Sätzen / Fragen
  - **Extraktion** von Entitäten / Fakten
- Schwierig zu extrahieren, aktuell oft nur rudimentär möglich.

Sprache zu Bedeutung?

Is the Eiffeltower higher than Notre Dame ?



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



Die Überführung natürlicher Sprache in eine formale Form ist bis heute ein **nicht abschließend gelöstes** Problem.

Beispiel: Das Projekt **Read The Web**[1]

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

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

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

Normalisierung

Erster Schritt in NLP-Systemen ist häufig die **Normalisierung**.
Diese beinhaltet in der Regel zwei Schritte:

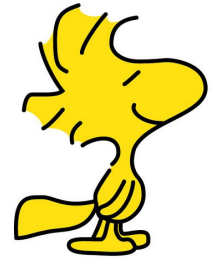
1. Segmentierung

   - Segmentierung von Texten in **Sätze**
   - Segmentierung von Sätzen in **Wörter/Terme/Tokens**
     (“house”, “rock ’n roll”, “#nlproc”)
   - häufig mit Hilfe **regulärer Ausdrücke**

2. Normalisierung von Wortformaten

   - häufig Grundformreduktion
   - **Lemmatisierung**, **Stemming**

Reguläre Ausdrücke



- ... bilden eine formale Sprache zur Spezifikation von Strings
- Buchstaben in eckigen Klammern = Disjunktion

[Ww]ood	Wood / wood	**Wood**stock is small indeed.
[0123456789]	beliebige Ziffer	Beverly Hills **9**0210.
yours|mine	alternativ Pipe-Symbol	Brace **yours**elf.
[0 – 9]	beliebige Ziffer	–
[a-z]	Kleinbuchstaben	W**o**odstock is small indeed.

- Negation: [^X] bedeutet "nicht X".

[^A-Z]	kein Großbuchstabe	W**o**odstock.
[^sS]	weder s noch S	S**u**re you can.
a^b	a^b (keine Negation)	Sure you can.

- Weitere Optionen:

colou?r	optionale Zeichen	**color**s and **colour**s.
ye*ah	0 oder mehr Vorkommen	**yah**! **yeah**! **yeeeah**!
ye+ah	1 oder mehr Vorkommen	yah! **yeah**! **yeeeah**!
beg.n	alle Buchstaben außer \n	begn **begin began beg3n**
.*	beliebig viele Buchstaben	**abcdefghijklmnop**
yeah$	Ende des Strings	yeah yeah **yeah**

Reguläre Ausdrücke in Python

```
1  import re
2
3  text = "We watched the Sunset over the Castle on the Hill."
4
5  # find occurrences of The/the (as strings)
6  for x in re.findall("[Tt]he", text):
7      print(x)
8  > the
9  > the
10 > the
11
12 # find words starting with capital letter (as Match objects)
13 for x in re.finditer("([A-Z][a-zA-Z]*)[ .]", text):
14     print(x.group(1))
15 > We
16 > Sunset
17 > Castle
18 > Hill
```

Anwendungen (auch im QA)

- **Segmentierung**, regelbasierte **Erkennung** z.B. von Preisen
- **Features** für Machine Learning – Komponenten
  (Abkürzung gefunden? → evtl. Definitions-Frage)

Outline

Worte: Grundformreduktion

- Worte sind die kleinste **bedeutungstragende Einheit** einer Sprache.
- Wir unterscheiden zwischen der **Grundform** eines Wortes und verschiedenen Varianten.
- Oft möchten wir **Grundformen** von Worten bestimmen *(warum?)*

Grundformreduktion

- von **Flexionsformen** auf Grundform

  Hauses → *Haus*
  ging → *gehen*

- Von der **Deviationsform** auf die Grundform

  Bücherei → *Buch*

- Von **Komposita** auf die Dekompositionen

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

Grundformreduktion

Lemmatisierung

- ... ist die Reduktion von Flexionen auf die Grundform *(siehe oben, z.B.* `housing` → `house`*).*
- Eine einfachere Form von Lemmatization heißt **Stemming**: Hier wird das Wortende abgeschnitten (`housing` → `hous`).

Stemming-Ansätze

1. **Lexikalische Verfahren**
   - speichern die Grundformen in Dictionaries
   - **Nachteil**: Erstellung und Pflege des Lexikons
   - vor allem für Fragen mit **starker Flexion** *(z.B. Deutsch)*
2. **Regelbasierte Verfahren**
   - Verwende **Regeln** wird zur Änderung / Kürzung des Suffix
   - Beispiel: Der **Porter Stemmer** für Englisch [2] *(Anwendung von Regeln in 4 sequenziellen Schritten)*

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

Wortarten (Parts of Speech, POS)

Die vier wichtigsten **“offenen” Wortarten**

Nomen

- verweisen auf konkrete Dinge (`ship`),
  abstrakte Dinge (`bandwidth`), Tätigkeiten (`singing`), ...
- können von Determinern (`high bandwidth`)
  und Possessiven (`his ship`) begleitet werden.
- Spezialfall: **Eigennamen** (engl. *proper nouns*), z.B. Colorado

Verben

- verweisen auf **Aktivitäten** und **Handlungen**
- werden durch **Flektion** verändert: `eat`, `eats`, `ate`, `eaten`

Adjektive

- beschreiben Eigenschaften von **Dingen** (`red ship`)

Adverben

- beschreiben Eigenschaften von **Verben** (`he walked slowly`)

Wieviele Wortarten gibt es insgesamt in Englisch?

45 *(in der sogenannten* **Penn Treebank***)*

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

Part-of-Speech Tagging

Wortarten (in Englisch) sind (sehr) mehrdeutig

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

- Die Wortart der meisten Wörter sind eindeutig (86%), aber **häufige** Wörter sind **oft mehrdeutig** (`that`, `back`, `set`, ...).
- Englische Texte enthalten ca. **60% mehrdeutige Tokens**.

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

= **bestimme die Wortart** der Tokens eines Textes

- Interessant für **viele Anwendungen**: Erkennung von Named Entities, Erkennung der Satzstruktur *(Parsing)*, **Schlüsselwörter** in Fragen erkennen, ...
- 92% Genauigkeit durch eine einfache **Baseline**, die für jedes Wort die häufigste Wortklasse wählt
- 97% Genauigkeit mit **Machine Learning** *(später)*

POS-Tagging in Python

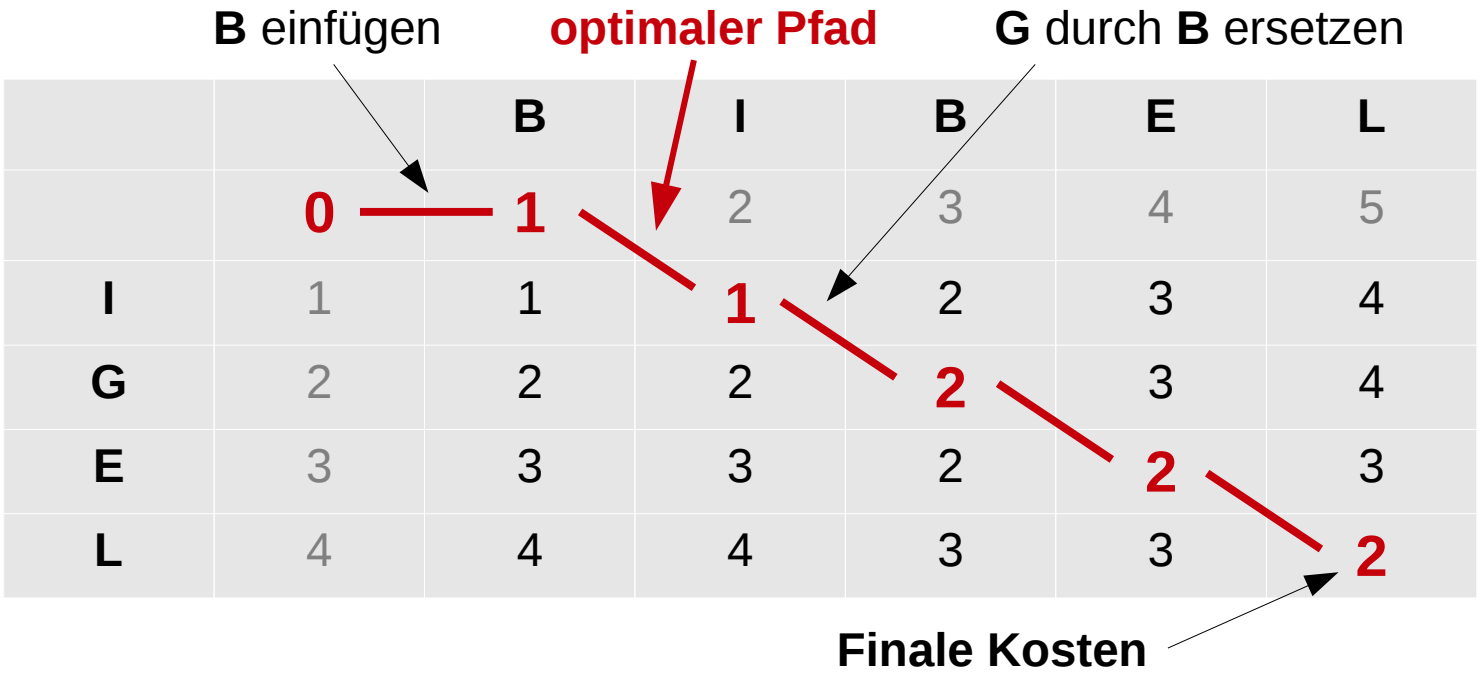
```
1  import nltk
2
3  text = "We watched the sunset over the castle on the hill."
4
5  tokens = nltk.word_tokenize(text)
6
7  result = nltk.pos_tag(tokens)
8
9  print(result)
10
11 > [('We', 'PRP'),              # OK (personal pronoun)
12    ('watched', 'VBD'),         # OK (verb past tense)
13    ('the', 'DT'),              # OK (determiner)
14    ('sunset', 'NN'),           # OK (noun singular)
15    ('over', 'IN'),             # OK (preposition)
16    ('the', 'DT'),              # OK (determiner)
17    ('castle', 'NN'),           # OK (noun singular)
18    ('on', 'IN'),               # OK (preposition)
19    ('the', 'DT'),              # OK (determiner)
20    ('hill', 'NN'),             # OK (noun singular)
21    ('.', '.')]
```

Syntaktische Wortähnlichkeit: Die Edit Distance

Wie vergleichen wir Terme syntaktisch (z.B. bei Tippfehlern)?

- **String Edit - Distanzen** (hier die sog. **Levenshtein-Distanz**)
- Gegeben zwei Terme/Strings $\mathbf{s}_1, \mathbf{s}_2$ der Länge n_1, n_2, berechnen wir die **Anzahl der Operationen** *(Einfügen, Entfernen, Ersetzen von Buchstaben)* um $\mathbf{s}_1$ in $\mathbf{s}_2$ zu **transformieren**.
- **Beispiel**: dist(`'typos'`, `'typohs'`) $= 1$
- **Beispiel**: dist(`'Weisbahdn'`, `'Wiesbaden'`) $= 4$
- **Auch für komplette Texte möglich**: dist(`'KI ist cool'`, `'K.I. ist kuhl'`) $= 5$
- Berechnung mittels **dynamischer Programmierung**

Levenshtein-Distanz: Algorithmus



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

- Erstelle eine $(n_1 \times n_2)$-**Kostenmatrix** D.
- Initialisiere die erste Spalte (Zeile) mit $0, 1, ..., n_1$ $(0, 1, ..., n_2)$.
- **Durchlaufe** die Matrix **zeilenweise**, und jede Zeile von **links nach rechts**. Berechne jede Zelle (i, j):

$$D_{i,j} \leftarrow min\Big(\underbrace{1 + D_{i-1,j}}_{Loeschen}, \underbrace{1 + D_{i,j-1}}_{Einfuegen}, \underbrace{\mathbf{1}_{s_1(i) \neq s_2(j)} + D_{i-1,j-1}}_{Ersetzen} \Big)$$

- D_{n_1,n_2} enthält am Ende die gesuchten Kosten.

Levenshtein-Distanz: Anmerkungen

- **Aufwand**? Anzahl der Matrix-Zellen $\rightarrow O(n_1 \cdot n_2)$
- In der Praxis suchen wir häufig den **ähnlichsten Term** aus dem **kompletten Vokabular**!
- Dies erfordert einen Vergleich des Eingabe-Terms s mit **allen** Termen des Vokabulars (**nicht realisierbar**!).
- Wir bestimmen deshalb eine kleine Auswahl an **Kandidaten** mittels **Vorfiltern** (sog. **n-Grammn-Filter**[2]).

Beispiel

- **Eingabestring**: `wiesbahdän`
  - `wi`: kommt vor in `wiesbaden`, `wiesel`, `wirtschaft`, ...
  - `ie`: kommt vor in `wiesbaden`, `wiesel`, `schief`, ...
  - `es`: kommt vor in `wiesbaden`, `wiesel`, `essen`, ...
  - `sb`: kommt vor in `wiesbaden`, `ausbilder`, ...
  - `..`: ..
  - `än`: kommt vor in `ändern`, `händler`
- `'wiesbaden'` ist ein Kandidat *(5 gemeinsame Bigramme)!*

[2] Manning, Raghavan, Schütze, „An Introduction to Information Retrieval“, 2009.

Outline

Natural Language Processing: Syntax Bild: [2]

Die **Syntax** einer Sprache wird durch **Grammatiken** definiert:

Konstituentengrammatik (Constituency Grammar)

- beschreiben Satz-Aufbau aus Komponenten
- **Beispiel**: *Noun Phrases (dt. Nominalphrase)* wie "Harry the Horse" / "they"
- Typischer Weise werden **kontextfreie Grammatiken** verwendet *(siehe AFS)*
- **Beispiel**: **siehe rechts**
- Deutlich mehr Regeln möglich *(z.B. 100 Verb-Typen [2] / Kap. 11.3)*

Noun	→	*flights* | *breeze* | *trip* | *morning*
Verb	→	*is* | *prefer* | *like* | *need* | *want* | *fly*
Adjective	→	*cheapest* | *non-stop* | *first* | *latest* | *other* | *direct*
Pronoun	→	*me* | *I* | *you* | *it*
Proper-Noun	→	*Alaska* | *Baltimore* | *Los Angeles* | *Chicago* | *United* | *American*
Determiner	→	*the* | *a* | *an* | *this* | *these* | *that*
Preposition	→	*from* | *to* | *on* | *near*
Conjunction	→	*and* | *or* | *but*

Grammar Rules			Examples
S	→	*NP VP*	I + want a morning flight
NP	→	*Pronoun*	I
	|	*Proper-Noun*	Los Angeles
	|	*Det Nominal*	a + flight
Nominal	→	*Nominal Noun*	morning + flight
	|	*Noun*	flights
VP	→	*Verb*	do
	|	*Verb NP*	want + a flight
	|	*Verb NP PP*	leave + Boston + in the morning
	|	*Verb PP*	leaving + on Thursday
PP	→	*Preposition NP*	from + Los Angeles

Die Penn Treebank

Was befindet sich in einer 'Treebank'?

1. Ein Text-Korpus
2. Eine Grammatik *(händisch definiert)*
3. Syntaktische Annotationen *(d.h., Syntaxbäume)* für sämtliche Sätze des Korpus *(händisch erfasst)*

Die Penn Treebank

- ... die bekannteste *(und historisch erste große)* Treebank
- Wichtige Grundlage der Computerlinguistik
  (Grundlage lernender Methoden für Parsing, POS-Tagging, ...)
- Syntaxbäume sind ein entscheidender Schritt zum automatisierten "Textverstehen"
- **Beispiel**: Autor-Werk-Relation im folgenden Satz:

> *What book was written by Bram Stoker in 1897?*

Die Penn Treebank (cont'd)

Penn Treebank Grammatik: Auszug

Regel	**Beispiel**
VP → VBD PP	`went to church`
VP → VBD PP PP	`went to church on Sunday`
...	...

Wieviele solcher Grammatik-Regeln enthält die Penn Treebank?

- Ungefähr $17,500$ (!)
- Die Regeln sind bis zu einem gewissen Grad redundant.

Parsing: Der CKY-Algorithmus

Gegeben einen Satz: Wie ermitteln wir den Syntax-Baum?

- Eine gängige Lösung: Der **Cocke-Kasami-Younger (CKY)-Algorithmus**
- (Auch) dieses Verfahren basiert auf **dynamischer Programmierung**
- Aufwand (für einen Satz mit n Tokens) $= O(n^2)$

Vorverarbeitung

- Der CKY-Algorithmus benötigt eine Grammatik in **Chomsky-Normalform**.
- Dies bedeutet: Es sind nur zwei Formen von Regeln erlaubt:
  $A \rightarrow BC$ (Ersetzung durch exakt zwei Nicht-Terminal-Symbole ...)
  $A \rightarrow a$ (... oder genau ein Terminal-Symbol)

Parsing: Der CKY-Algorithmus

Vorverarbeitung (cont'd)

Wir können eine gegebene Grammatik mit **drei einfachen Tricks** in Chomsky-Normalform überführen:

1. Nur ein NT-Symbol auf der rechten Seite? **Regel ersetzen**.

$$A \to B,\ B \to X \quad \Rightarrow \quad A \to X$$

2. Gemischte T- und NT-Symbole rechts? Führe für x ein **Brückensymbol** X ein.

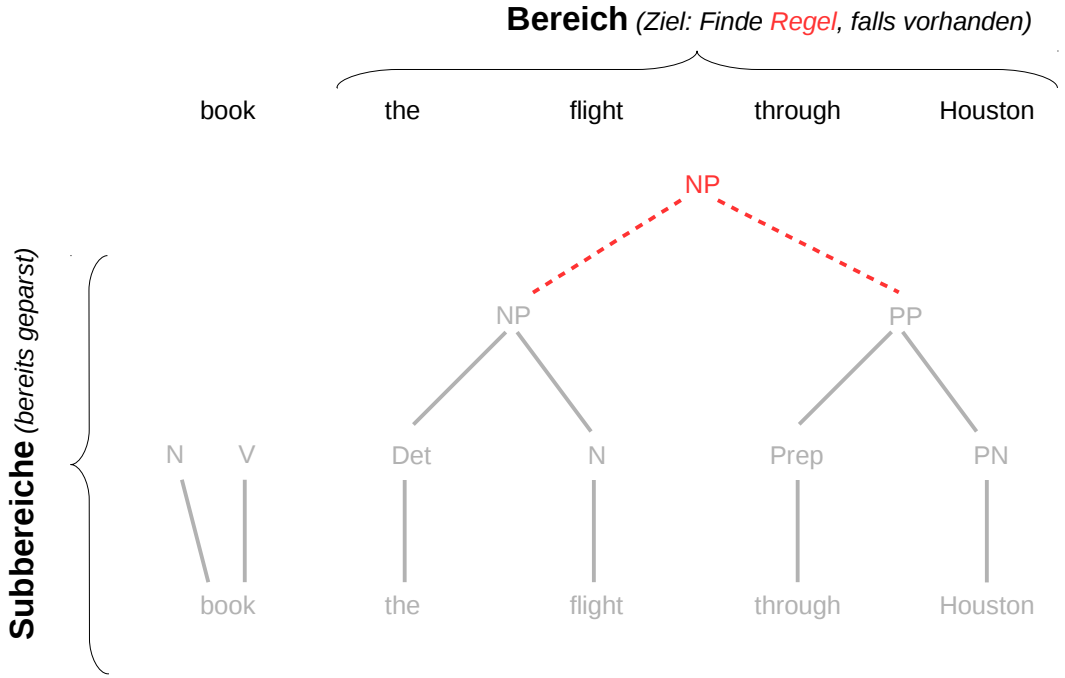
$$A \to B\,x \quad \Rightarrow \quad A \to B\,X,\ X \to x$$

3. Zu viele NT-Symbole rechts? Führe **Brückensymbol** Y ein.

$$A \to B\,C\,D \quad \Rightarrow \quad A \to Y\,D,\ Y \to B\,C$$

Parsing: Der CKY-Algorithmus

- Der Algorithmus arbeitet **bottom-up**, von den **einzelnen Termen** hin zum **ganzen Satz**.
- Gegeben sei ein Subbereich der Länge $k > 1$. Dann ist dieser Subbereich gemäß einer Regel $A \rightarrow B\ C$ in **zwei Teile** zu zerlegen.
- Für sämtliche **kleineren Subbereiche** der Länge $< k$ **kennen** wir bereits den **Subbaum**.

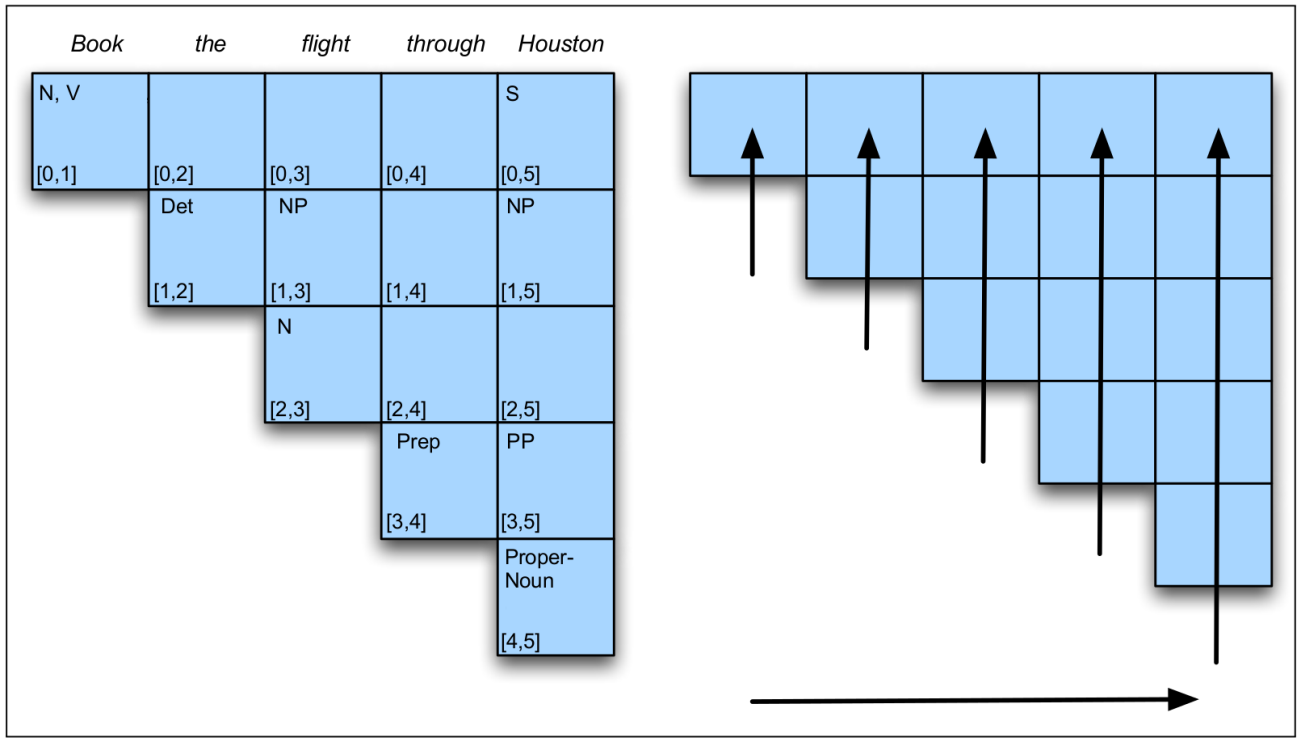


(Mini-) Grammatik

V → book
N → book
Det → the
N → flight
Prep → through
PN → Houston

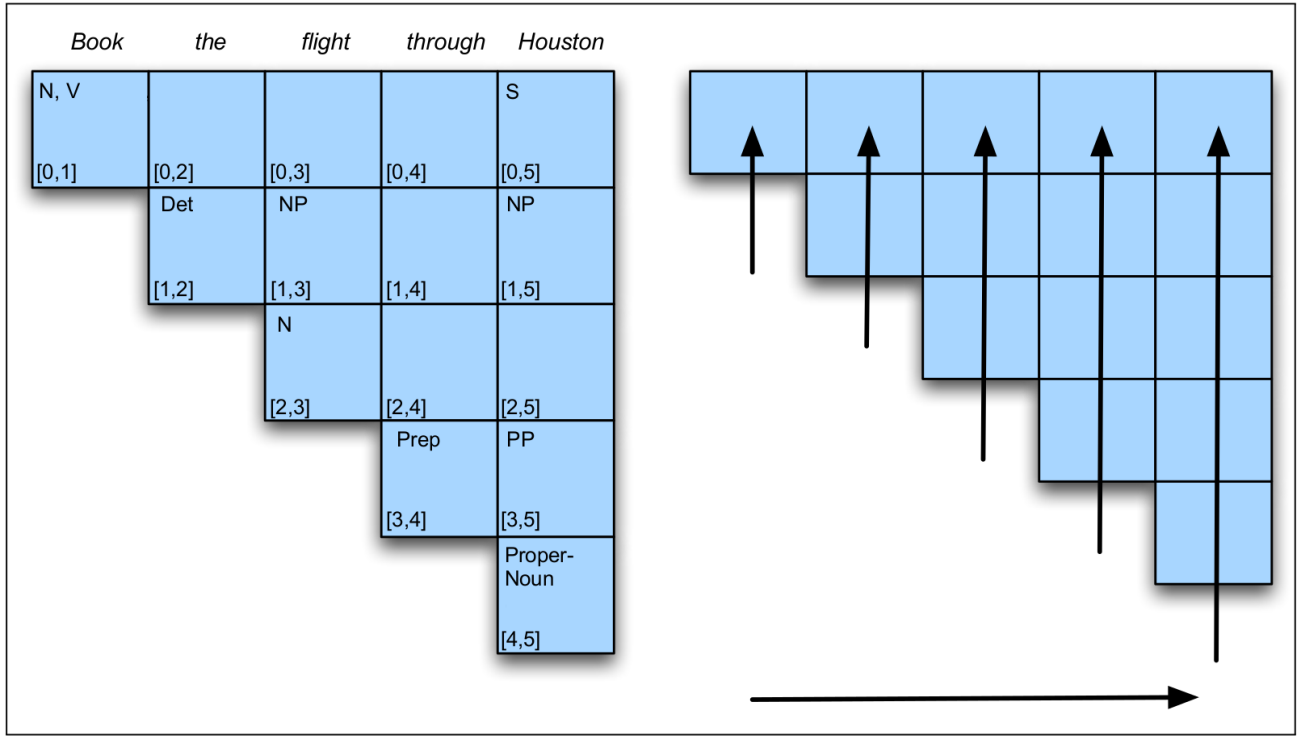
PP → Prep PN
NP → Det N
NP → NP PP
S → V NP

Parsing: Der CKY-Algorithmus Bild: [2]



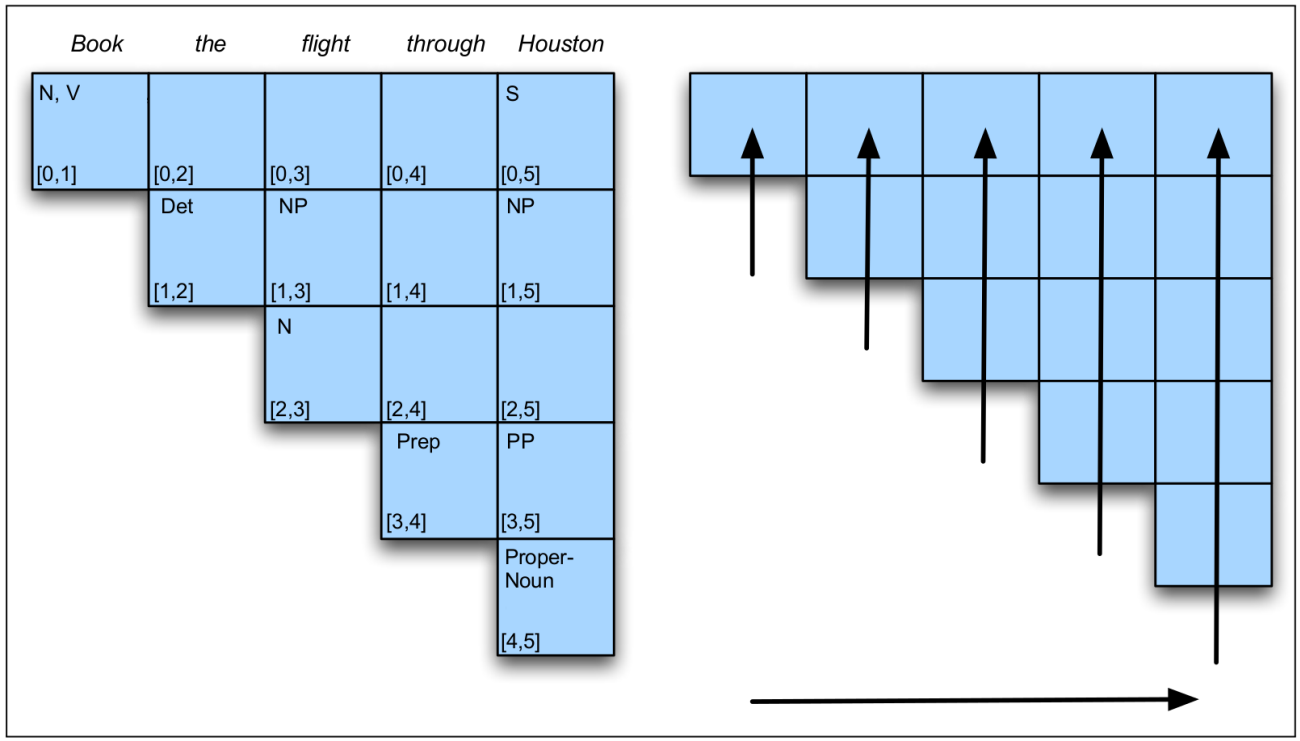
- Wir ordnen sämtliche **Subbereiche** (i, j) *(inkl. Term i bis exkl. Term j)* in einer **Matrix** an.
- Initialisierung mit Einzeltermen / **Terminalsymbolen** *(→ Diagonale)*

Parsing: Der CKY-Algorithmus Bild: [2]



- Wir durchlaufen die Matrix **spaltenweise** und die Spalten **von unten nach oben**
- An jeder Position (i, j) prüfen wir für alle $k = i + 1, ..., j$, ob wir die zwei Subbereiche (i, k) und (k, j) mit einer **Regel** verbinden können.

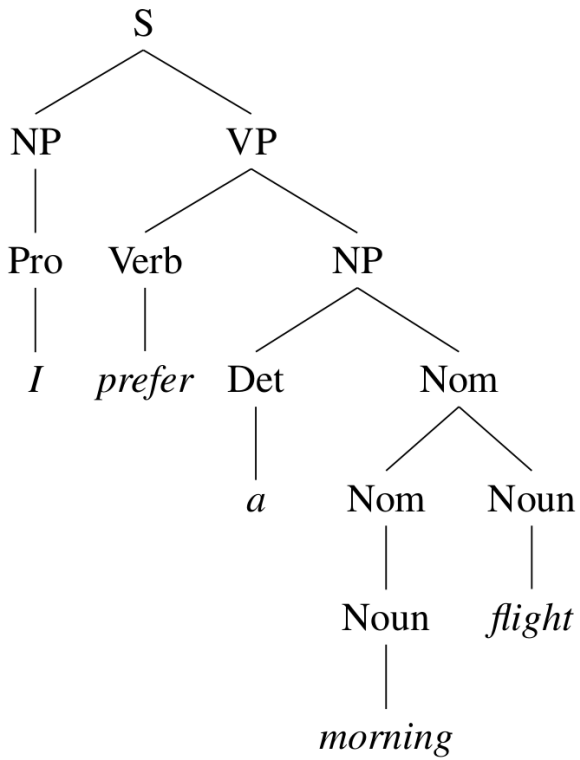
Parsing: Der CKY-Algorithmus Bild: [2]

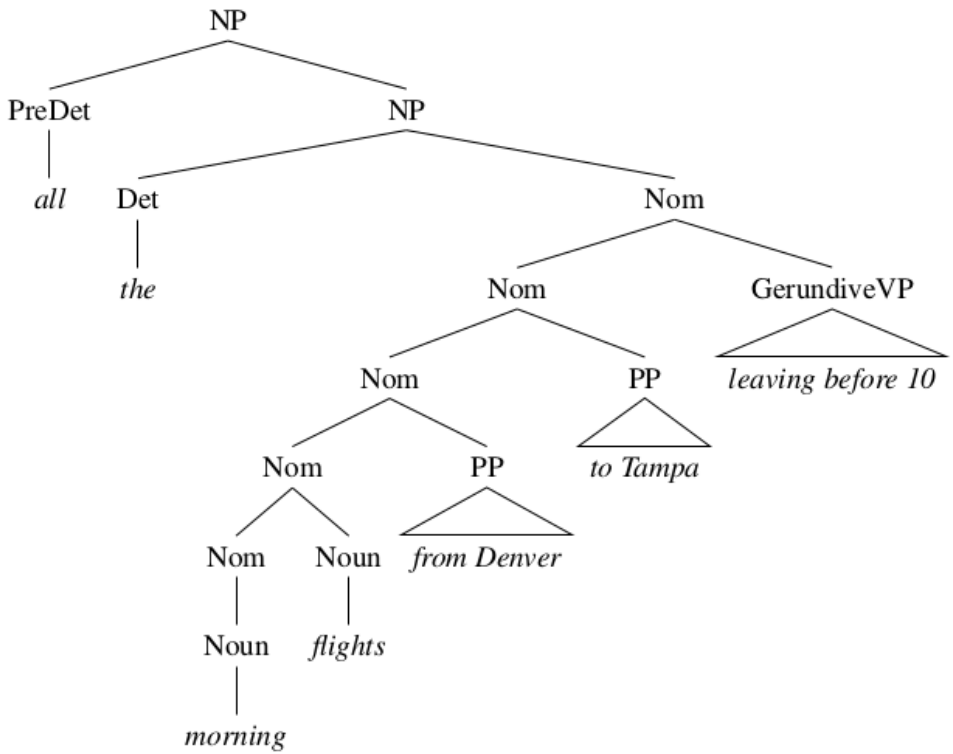


- Am Ende steht der Parse des Satzes in $(0, n)$.

Parsing (cont'd) Bild: [2]

- Mit der Grammatik können wir einen Satz **herleiten**. Sätze die wir nicht herleiten können nennen wir **"ungrammatisch"**.

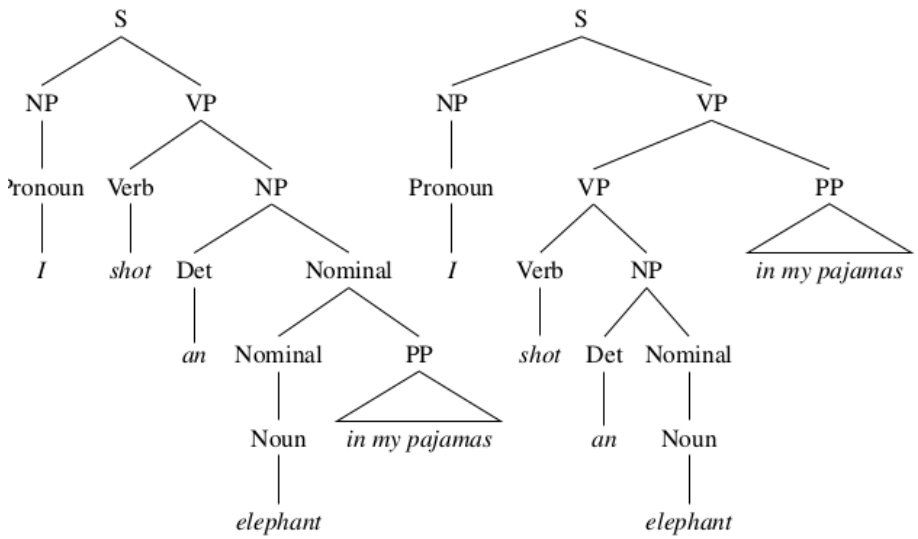




Parsing (cont'd) Bild: [2]

"One morning I shot an elephant in my pajamas. How he got into my pajamas I don't know."

(Groucho Marx, Animal Crackers, 1930)



Syntax ist mehrdeutig

- Disambiguation ist ein **schwieriges Problem**: Wir benötigen statistische, semantische, und Kontext-Information.
- Typisch: Probabilistisches Parsing mit Machine Learning [2]

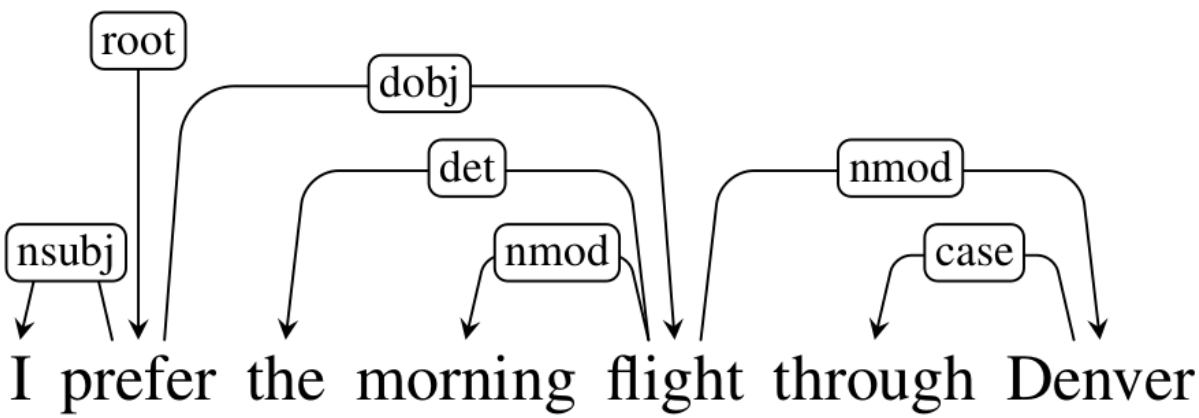
Grammatik II: Dependenzparser

Dependenzparser *(engl. Dependency Parsers)* **verzichten** komplett auf kontextfreie Grammatiken:

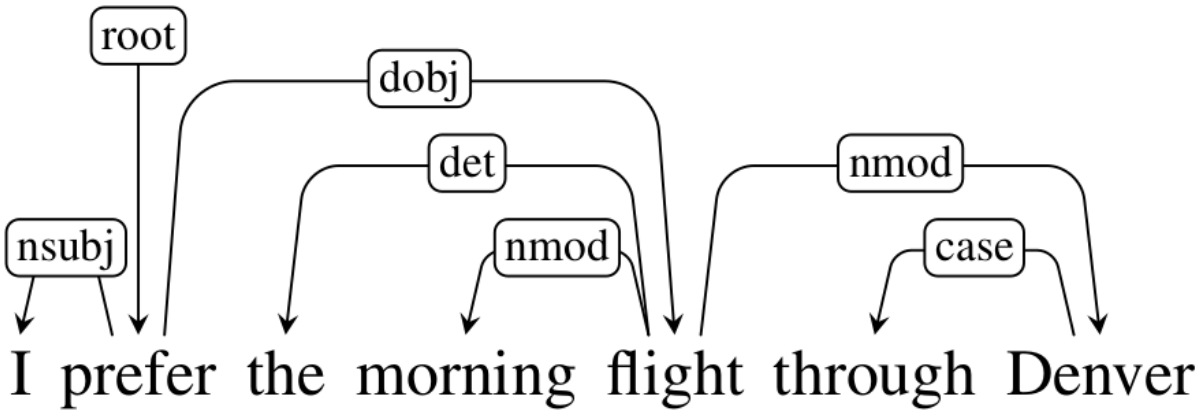
Ansatz

- **Konstituentenparser**: beschäftigen sich mit dem **(rekursiven) Aufbau** eines Ansatzes aus **Komponenten**.
- **Dependenzparser**: Wir beschreiben einen Satz über die **Relationen / Abhängigkeiten** zwischen seinen Worten.
- Relationen zwischen **Worten**, **nicht Konstituenten**!

Beispiel



Grammatik II: Dependenzparser



- Eine **Abhängigkeit** besteht zwischen einem **regierenden Wort** (engl. *head*) und einem **abhängigen Wort** (engl. *dependent*).
- Es entsteht der sog. **Dependenzbaum**, mit dem **Prädikat** des Satzes als **Wurzel**.
- Prädikate ("prefer") sind direkt mit Subjekt ("I") und Objekt ("flight") verbunden → nützlich z.B. für Question Answering.
- Subjekt und Objekt sind ihrerseits wieder mit sog. **Modifikatoren** verbunden (morning flight).
- Die **Wort-Reihenfolge** wird weg-abstrahiert!

Dependenzparser: Datenquellen

Ressourcen

- Die verschiedenen Typen von Dependenz-Relationen sind z.B. im **Universal Dependencies project** definiert.
- Zielsetzung: Relationen, die **linguistisch motiviert** und **nützlich** sind.

Relation	Examples with *head* and **dependent**
NSUBJ	**United** *canceled* the flight.
DOBJ	United *diverted* the **flight** to Reno.
	We *booked* her the first **flight** to Miami.
IOBJ	We *booked* **her** the flight to Miami.

- Für die Penn Treebank sind **Dependency-Annotationen** verfügbar.
- Auch hier gibt es automatische Parser *(z.B. Shift-Reduce Parsing, [2], Kapitel 14.4)*.

Outline

Term-Ähnlichkeiten

- Für zahlreiche Anwendungen *(Information Retrieval, Question Answering, Dokument-Clustering, ...)* ist es von Interesse, **Ähnlichkeiten** zwischen Termen/Phrasen zu bestimmen.
- Bsp. **Query Expansion**: Falls der Query `'laptop'` nichts findet, fügen wir dem Query das Synonym `'notebook'` hinzu.

Herausforderungen

- Sprache ist dynamisch → Pflege eines Thesaurus aufwändig.
- "Ähnlichkeit" hängt von Anwendung und Kontext ab.

`'laptop'` → `'notebook'` ✓
`'burning smartphone'` → `'samsung'` ↯

Ansatz: Lernen von Cooccurence-Statistiken

- **Lernen** von Term-Ähnlichkeiten auf Basis eines **Textkorpus**.
- Terme sind ähnlich, wenn Sie *ähnlich auftreten*.

Term-Ähnlichkeiten

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

Beispiel: Was wissen wir über "tesgüino"?

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

Zwei Arten von Term-Ähnlichkeit

- **1. Ordnung** ("Kollokationen"): w_2 ist ähnlich zu w_1, wenn w_2 häufig **im Kontext** von w_1 auftritt *(Bsp.* `wrote` → `book`*)*.
- **2. Ordnung**: w_2 ist ähnlich zu w_1, wenn w_1 und w_2 häufig in **ähnlichen Kontexten** auftreten *(Bsp.* `vodka` → `whisky`*)*.
- Der "Kontext" ist hier üblicher Weise ein lokales Fenster von $\pm(1-8)$ Termen.

Term-Ähnlichkeiten

- **Gegeben**: Vokabular von n Termen/Phrasen + Textkorpus.
- **Ziel**: Berechnung eine $n \times n$-Matrix A (**Term-Term-Matrix**), so dass A_{ij} die Ähnlichkeit zwischen Term w_i und Term w_j ist.
- Einfache Möglichkeit:

$$A_{ij} := \#\text{ Vorkommen von } w_j \text{ in der Umgebung von } w_i$$

- Warum ist dieses Ähnlichkeitsmaß **nicht optimal**?

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

PPMI-Termähnlichkeit

Positive Pointwise Mutual Information (PPMI) ([2], Kapitel 15)

Es sei w_i ein Term und w_j ein "Kontext-Term", und A die Term-Term-Matrix mit den Häufigkeiten von Term-Paaren (s.o.). Wir berechnen

- Die Wahrscheinlichkeit, dass w_i in einem **zufälligen lokalen Term-Paar** auftritt:

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

- Die Wahrscheinlichkeit, dass w_j als **Kontext-Term** auftritt:

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

- Die Wahrscheinlichkeit, dass **beide Terme** in einem Kontext auftreten:

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

PPMI-Termähnlichkeit

- Wir wissen *(siehe Statistik)*: Sind die Terme **unabhängig**, gilt:

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

- Uns interessieren Paare von Termen, die **ungewöhnlich häufig gemeinsam** auftreten, d.h. es gilt:

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

- Dies führt zur sogenannten **Pointwise Mutual Information (PMI)**:

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

PPMI-Termähnlichkeit

- Je **höher** die PMI, desto **ähnlicher** zwei Terme
- Beispiel: Die Top-10 und Bottom-10 PMI-Paare von Wikipedia

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

Von der PMI zur PPMI

- Da uns negative Werte nicht interessieren *(und häufig Ausreißer enthalten)*, clippen wir sie auf 0 und erhalten die **Positive Pointwise Mutual Information (PPMI)**:

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

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

PPMI-Termähnlichkeit

Anmerkungen

- **Problem**: Sehr seltene Worte, die zufällig 2-3 mal häufiger als üblich im Kontext auftreten, erhalten manchmal **extreme PPMI-Werte**. Deshalb **glättet** man gerne die Term-Term-Matrix A vor der PPMI-Berechnung, indem man eine Konstante (z.B. 1-2) dazu addiert.
- PPMI ist ein Maß für Ähnlichkeit *erster Ordnung*, wir können hieraus aber auch Ähnlichkeiten *zweiter Ordnung* ableiten [1].

References I

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

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