Mittwoch: Parser und Tagger

6 Parser und Tagger
- TreeTagger
- MaltParser
- Stanford Parser

Parser und Tagger

TreeTagger

- Statistischer Part-of-Speech-Tagger
- Frei verfügbar
- `http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/`
- Viele Erweiterungen
- Auf unseren Servern unter `/resources/processors/tagger/tree-tagger`
- Trainierte Modelle für Englisch, Deutsch, Französisch, Niederländisch, Griechisch, Italienisch, ...

Verzeichnisinhalt

- `bin` – Die eigentlichen Programme
- `cmd` – Shell-Skripte mit eingebautem Preprocessing
- `doc` – Papers
- `lib` – Gespeicherte Modelle

Englisches Tagset

NNS	Noun, plural
NN	Noun, singular or mass
DT	Determiner
JJ	Adjective
MD	Modal verb
VBP	Present tense verb

Vollständige Liste:
`http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/data/Penn-Treebank-Tagset.pdf`

Benutzung

- TreeTagger stellt Voreinstellungen (in Skripten) für die verschiedenen Sprachen bereit:
  `tree-tagger-english` , **`tree-tagger-german`** , ...
- Um Feineinstellungen vorzunehmen, muss `tree-tagger` direkt aufgerufen werden: `tree-tagger`
- Einstellungen:
  `tree-tagger [OPTIONS] <parameter file> [<input file> [<output file>]]`
- Options:
  - `-token` Gibt das Token aus
  - `-lemma` Gibt das Lemma aus
  - `-prob` Gibt die Wahrscheinlichkeit aus
  - `-lex f` Liest ein zusätzliches Lexikon aus Datei `f`

Benutzung

- TreeTagger stellt Voreinstellungen (in Skripten) für die verschiedenen Sprachen bereit:
  `tree-tagger-english` , **`tree-tagger-german`** , ...
- Um Feineinstellungen vorzunehmen, muss **`tree-tagger`** direkt aufgerufen werden: **`tree-tagger`**
- Einstellungen:
  `tree-tagger [OPTIONS] <parameter file> [<input file> [<output file>]]`
- Options:
  - `-token` Gibt das Token aus
  - `-lemma` Gibt das Lemma aus
  - `-prob` Gibt die Wahrscheinlichkeit aus
  - `-lex f` Liest ein zusätzliches Lexikon aus Datei `f`

Benutzung

- TreeTagger stellt Voreinstellungen (in Skripten) für die verschiedenen Sprachen bereit:
  `tree-tagger-english` , **`tree-tagger-german`** , ...
- Um Feineinstellungen vorzunehmen, muss **`tree-tagger`** direkt aufgerufen werden: **`tree-tagger`**
- Einstellungen:
  `tree-tagger [OPTIONS] <parameter file> [<input file> [<output file>]]`
- Options:
  - `-token` Gibt das Token aus
  - `-lemma` Gibt das Lemma aus
  - `-prob` Gibt die Wahrscheinlichkeit aus
  - `-lex f` Liest ein zusätzliches Lexikon aus Datei `f`

Benutzung

- TreeTagger stellt Voreinstellungen (in Skripten) für die verschiedenen Sprachen bereit:
  `tree-tagger-english` , **`tree-tagger-german`** , ...
- Um Feineinstellungen vorzunehmen, muss **`tree-tagger`** direkt aufgerufen werden: **`tree-tagger`**
- Einstellungen:
  `tree-tagger [OPTIONS] <parameter file> [<input file> [<output file>]]`
- Options:

  `-token` Gibt das Token aus
  `-lemma` Gibt das Lemma aus
  `-prob` Gibt die Wahrscheinlichkeit aus
  `-lex f` Liest ein zusätzliches Lexikon aus Datei `f`

Benutzung

- TreeTagger stellt Voreinstellungen (in Skripten) für die verschiedenen Sprachen bereit:
  `tree-tagger-english` , **`tree-tagger-german`** , ...
- Um Feineinstellungen vorzunehmen, muss **`tree-tagger`** direkt aufgerufen werden: **`tree-tagger`**
- Einstellungen:
  `tree-tagger [OPTIONS] <parameter file> [<input file> [<output file>]]`
- Options:

  `-token` Gibt das Token aus
  `-lemma` Gibt das Lemma aus
  `-prob` Gibt die Wahrscheinlichkeit aus
  `-lex f` Liest ein zusätzliches Lexikon aus Datei `f`

Benutzung

- TreeTagger stellt Voreinstellungen (in Skripten) für die verschiedenen Sprachen bereit:
  `tree-tagger-english` , **`tree-tagger-german`** , ...
- Um Feineinstellungen vorzunehmen, muss **`tree-tagger`** direkt aufgerufen werden: **`tree-tagger`**
- Einstellungen:
  `tree-tagger [OPTIONS] <parameter file> [<input file> [<output file>]]`
- Options:

  `-token` Gibt das Token aus
  `-lemma` Gibt das Lemma aus
  `-prob` Gibt die Wahrscheinlichkeit aus
  `-lex f` Liest ein zusätzliches Lexikon aus Datei `f`

Benutzung

- TreeTagger stellt Voreinstellungen (in Skripten) für die verschiedenen Sprachen bereit:
  `tree-tagger-english` , **`tree-tagger-german`** , ...
- Um Feineinstellungen vorzunehmen, muss **`tree-tagger`** direkt aufgerufen werden: **`tree-tagger`**
- Einstellungen:
  `tree-tagger [OPTIONS] <parameter file> [<input file> [<output file>]]`
- Options:

  `-token` Gibt das Token aus
  `-lemma` Gibt das Lemma aus
  `-prob` Gibt die Wahrscheinlichkeit aus
  `-lex f` Liest ein zusätzliches Lexikon aus Datei `f`

Benutzung

- TreeTagger stellt Voreinstellungen (in Skripten) für die verschiedenen Sprachen bereit:
  `tree-tagger-english` , **`tree-tagger-german`** , ...
- Um Feineinstellungen vorzunehmen, muss **`tree-tagger`** direkt aufgerufen werden: **`tree-tagger`**
- Einstellungen:
  `tree-tagger [OPTIONS] <parameter file> [<input file> [<output file>]]`
- Options:

  `-token` Gibt das Token aus
  `-lemma` Gibt das Lemma aus
  `-prob` Gibt die Wahrscheinlichkeit aus
  `-lex f` Liest ein zusätzliches Lexikon aus Datei **`f`**

Ein- und Ausgabe

`tree-tagger-(english|german|…)`

Der Text muss praktisch nicht vorverarbeitet sein.

`tree-tagger`

Ein Token pro Zeile.

`tree-tagger-german`: Ausgabe

```
      reading parameters ...
      tagging ...
      finished.
Die     ART     d
Sonne   NN      Sonne
scheint VVFIN   scheinen
.       $.      .
```

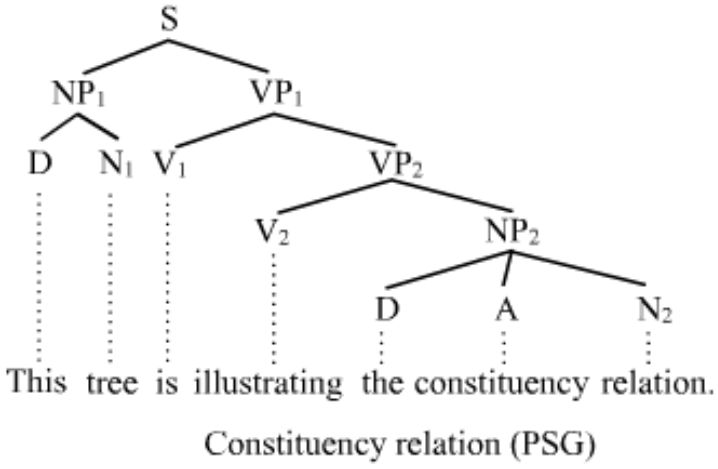
Übung 8

Parser und Tagger

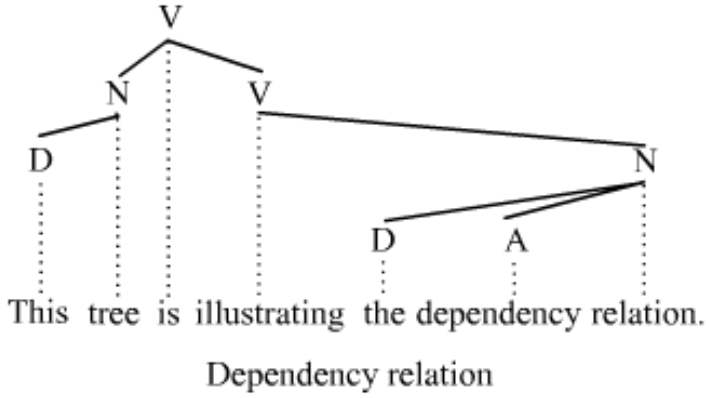
MaltParser

- Deterministischer Dependenz-Parser, implementiert in Java (seit Version 1.0)
- Parsing-Modell wird aus einer Treebank induziert
- Neue Daten werden mit dem induzierten Modell geparst
- Webseite: `http://maltparser.org/download.html`
- Auf unseren Servern unter `/resources/processors/parser/maltparser-1.8/`
- Modelle für Deutsch, Englisch, Französisch und Schwedisch
- Eingabe und Ausgabe im *CoNLL Format*

Dependenz- vs. Konstituenten-Parser



Constituency relation (PSG)

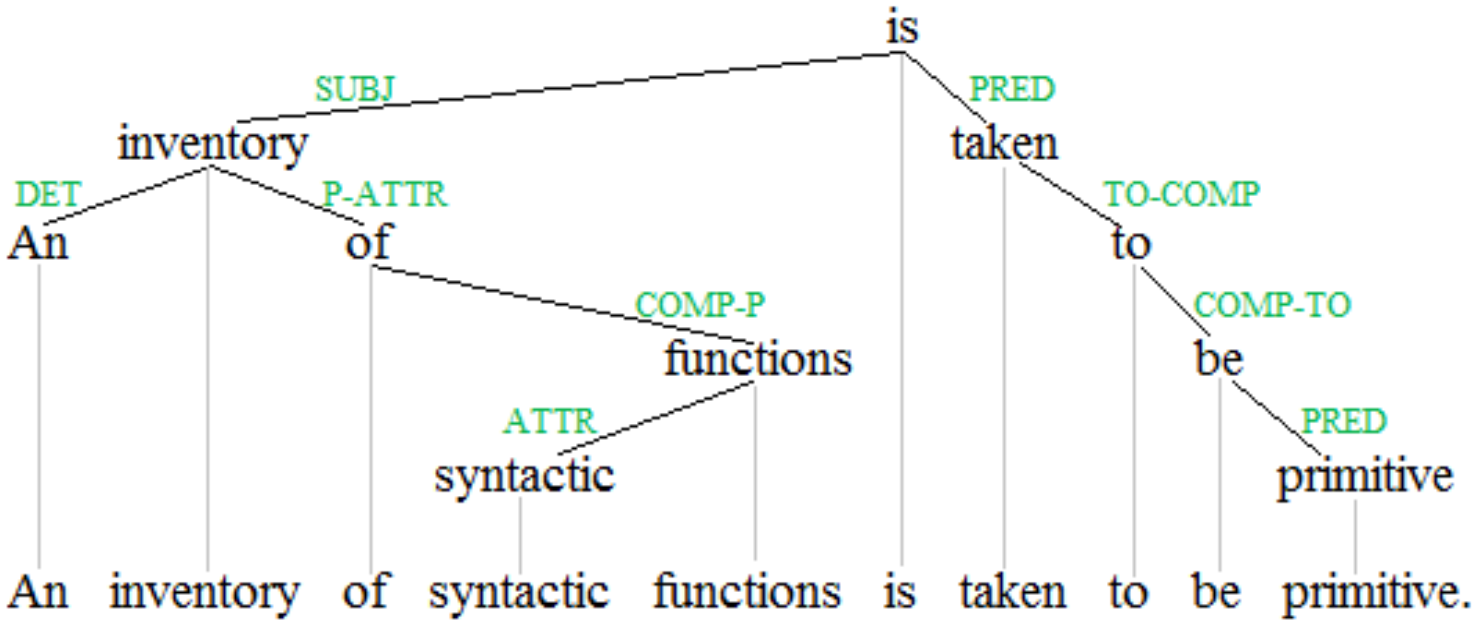


Dependency relation

Quelle:

`https://en.wikipedia.org/wiki/Phrase_structure_grammar`

Dependenzbäume mit Kantenbeschriftungen



Quelle: ByTjo3ya-Ownwork,CCBY-SA3.0,https://commons.wikimedia.org/w/index.php?curid=21976793

CoNLL Format

- Standardisiertes Format für die Eingaben/Ausgaben bei CoNLL Shared Task 2007
- Text-Dateien mit der Erweiterung `.conll`
- Eine Zeile besteht aus 10 durch `TAB` getrennten Feldern
- Ein Satz besteht aus einem oder mehreren Tokens, je eines auf einer Zeile
- Eine Datei kann mehrere Sätze enthalten (getrennt durch eine Leerzeile)
- Leerzeichen in den Feldern sind nicht erlaubt!

`http://ilk.uvt.nl/conll/#dataformat`

CoNLL Format - Felder

1. `ID` - Der Zähler für Tokens (beginnt mit 1 in jedem Satz)
2. `FORM` - Wortform
3. `LEMMA` - Lemma des Wortes
4. `CPOSTAG` - POS-Tag (sprachabhängig)
5. `POSTAG` - Detailliertes POS-Tag (sprachabhängig)
6. `FEATS` - Syntaktische und/oder morphologische Merkmale (sprachabhängig)
7. `HEAD` - Head-Token für das aktuelle Token (ID oder 0)
8. `DEPREL` - Dependenzrelation mit Head-Token (von den Treebank-Annotationen abhängig)
9. `PHEAD` - Projektive Head-Token für das aktuelle Token (ID oder 0)
10. `PDEPREL` - Dependenzrelation mit Head-Token (von den Treebank-Annotationen abhängig)

CoNLL Format - Bemerkungen

- Die folgende Felder *müssen* immer ausgefüllt werden:
  - Bei den Trainingsdaten:
    `ID`, `FORM`, `CPOSTAG`, `POSTAG`, `HEAD`, `DEPREL`
  - Bei den Testdaten:
    `ID`, `FORM`, `CPOSTAG`, `POSTAG`
- Andere Felder können mit dem Platzhalter „_" besetzt werden
- Falls es keine detaillierten POS-Tags für eine bestimmte Sprache gibt, sind die Felder `CPOSTAG` und `POSTAG` gleich
- Mehrere Merkmale werden durch das Symbol „|" getrennt
- Dependenzrelation für den Startknoten ist `ROOT` (mit dem `HEAD`-Wert 0)

CoNLL Format - Beispiel

Satz aus einem Trainingsset

1	Trouble	_	NN	NN	_	2	SBJ	_	_
2	is	_	VBZ	VBZ	_	0	ROOT	_	_
3	,	_	,	,	_	2	P	_	_
4	she	_	PRP	PRP	_	5	SBJ	_	_
5	has	_	VBZ	VBZ	_	2	PRD	_	_
6	lost	_	VBN	VBN	_	5	VC	_	_
7	it	_	PRP	PRP	_	6	OBJ	_	_
8	just	_	RB	RB	_	6	MNR	_	_
9	as	_	RB	RB	_	8	AMOD	_	_
10	quickly	_	RB	RB	_	8	AMOD	_	_
11	.	_	.	.	_	2	P	_	_

CoNLL Format - Beispiel

Satz aus einem Testset

1	Pierre	_	NNP	NNP	_
2	Vinken	_	NNP	NNP	_
3	,	_	,	,	_
4	61	_	CD	CD	_
5	years	_	NNS	NNS	_
6	old	_	JJ	JJ	_
7	,	_	,	,	_
8	will	_	MD	MD	_
9	join	_	VB	VB	_
10	the	_	DT	DT	_
11	board	_	NN	NN	_
12	as	_	IN	IN	_
13	a	_	DT	DT	_
14	nonexecutive	_	JJ	JJ	_
15	director	_	NN	NN	_
16	Nov.	_	NNP	NNP	_
17	29	_	CD	CD	_
18	.	_	.	.	_

CoNLL Format - Beispiel

Andere Sprache

1	Grundavdraget	_	N	NN	DD|SS	2	SS	_	_
2	blir	_	V	BV	PS	0	ROOT	_	_
3	2500	_	R	RO	_	4	DT	_	_
4	kr	_	N	NN	_	2	SP	_	_
5	(	_	I	IR	_	0	ROOT	_	_
6	=	_	I	IG	_	15	DT	_	_
7	4500	_	R	RO	_	8	DT	_	_
8	minskat	_	P	TP	PA	0	ROOT	_	_
9	med	_	PR	PR	_	14	AA	_	_
10	1/5	_	N	NN	_	9	PA	_	_
11	av	_	PR	PR	_	10	ET	_	_
12	10000	_	R	RO	_	13	DT	_	_
13	kr	_	N	NN	_	11	PA	_	_
14	)	_	I	IR	_	0	ROOT	_	_
15	.	_	P	IP	_	2	IP	_	_

Erster Start des MaltParsers

- Starten des MaltParsers ohne Argumente:

```
:~$ java -jar /path/to/maltparser-1.8/maltparser-1.8.jar
```

  - Gibt die Hauptoptionen für den MaltParser aus

- Starten der Hilfe für den MaltParser:

```
:~$ java -jar /path/to/maltparser-1.8/maltparser-1.8.jar -h
```

  - Gibt alle Optionen für den MaltParser aus

Den MaltParser trainieren

```
:~$ java -jar /path/to/maltparser-1.8/maltparser-1.8.jar -c model \
-i training-file.conll -m learn
```

- **`model`** : Name des Modells
  Achtung: nur den Namen ohne Pfad angegeben, die Erweiterung (**`.mco`**) wird *automatisch* angefügt!
- **`training-file.conll`** : Datei mit den Trainingsdaten im CoNLL-Format
- **`-m learn`** : der Parser wird im Lern-Modus gestartet

Das erstellte Modell wird (hier) in einer Datei namens **`model.mco`** im aktuellen Verzeichnis gespeichert.

Vortrainierte Modelle für den MaltParser

- Für diejenigen, die nicht mit unterschiedlichen Algorithmen und Modellen experimentieren möchten, sondern nur einen Dependenzparser brauchen
- Aktuell sind drei fertige Modelle vorhanden: **`engmalt`**, **`fremalt`** und **`swemalt`** (alle jeweils **`*.linear`** und **`*.poly`**)
- Download: `http://maltparser.org/mco/mco.html`
- Basieren auf Penn Treebank, French Treebank und Swedisch Treebank

Parsen von Daten im MaltParser

```
:~$ java -jar /path/to/maltparser-1.8/maltparser-1.8.jar -c model\
-i test-file.conll -o result-file.conll -m parse
```

- **`model`** : Name des Modells
  Achtung: nur den Namen ohne die Erweiterung (**`.mco`**) angeben
- **`test-file.conll`** : Datei mit den Testdaten
- **`result-file.conll`** : Datei mit den Ergebnissen des Parsers
- **`-m parse`** : der Parser wird im Parsing-Modus gestartet

MaltParser – Weitere Informationen

- Userguide: `http://maltparser.org/userguide.html`
- (Technische) Informationen zur Optimierung:
  `http://www.maltparser.org/guides/opt/quick-opt.pdf`

Übung 9

Parser und Tagger

Stanford Parser

- Java-Implementierung eines probabilistischen NLP-Parser
- Kann sowohl Dependenzstrukturen als auch Phrasenstrukturen ausgeben
- GNU General Public License
- Download: `http://nlp.stanford.edu/software/lex-parser.shtml`
- Auf unseren Servern unter `/resources/processors/parser/stanfordparser-3.6.0`
- Modelle für Englisch, Deutsch, Französisch, Chinesisch, Arabisch, Italienisch, Bulgarisch, Portugiesisch, ...
- Online-Version: `http://nlp.stanford.edu:8080/parser/`

Online Demo-Version

Siehe Browser.

Dependenzen im Stanford Parser

- Mehrere Formalismen verfügbar
- Alle Dependenzen sind binäre Relationen
- Tokens werden zusammen mit dem Index gezeigt
  Beispiel:
  `nsubj(makes-8, Bell-1)`
- Standard: *Universal Dependencies*
  - Insgesamt: 40 grammatische Funktionen
  - sprachunabhängig

Dependenztypen

- Universal Dependencies - alle Dependenzen werden individuell dargestellt
  Beispiel:
  `prep(based-7, in-8)`
  `pobj(in-8, LA-9)`
- Enhanced Universal Dependencies - u.a.: manche Dependenzen (z.B. mit Präpositionen, Konjunktionen) werden zusammengefasst.
  Beispiel:
  `prep:in(based-7, LA-9)`

Eingabe-Datei

- Einfache Textdatei
- Ein oder mehrere Sätze pro Zeile (keine Leerzeilen zwischen den Zeilen)
- Mehrere Sätze in einer Zeile werden mit einem Sentence Splitter getrennt

Beispiel:

```
The strongest rain ever recorded in India shut down
the financial hub of Mumbai, snapped communication
lines, closed airports and forced thousands of people
to sleep in their offices or walk home during the
night, officials said today.
```

Generierung der Ausgabe

```
:~$ java -mx200m edu.stanford.nlp.parser.lexparser.\
LexicalizedParser -outputFormat "wordsAndTags,penn,typedDependencies
"englishPCFG.ser.gz text
```

- **englishPCFG.ser.gz** : Vortrainiertes Modell für Englisch
- **-outputFormat** : Format für die Ausgabe (Mehrere Optionen durch Kommata getrennt)
- **text** : Eingabe-Datei mit den Sätzen
- **-sentences newline** : optional. Unterdrückt das Starten des Sentence Splitters

Das Ergebnis wird auf STDOUT ausgegeben.

Das Ausgabeformat

Die Optionen für `outputFormat` im Beispiel:

- `wordsAndTags`: Text mit POS-Tags
- `penn`: Syntaktische Baumstrukturen
- `typedDependencies`: Grammatische Relationen in Dependenzformat (Enhanced Universal Dependencies)

Ausgabe-Datei – Teil 1 und 2

```
The/DT strongest/JJS rain/NN ever/RB recorded/VBN in/IN
India/NNP shut/VBD down/RP the/DT financial/JJ hub/NN
of/IN Mumbai/NNP ,/, snapped/VBD communication/NN lines/NNS
,/, closed/VBD airports/NNS and/CC forced/VBD thousands/NNS
of/IN people/NNS to/TO sleep/VB in/IN their/PRP$ offices/NNS
or/CC walk/VB home/NN during/IN the/DT night/NN ,/,
officials/NNS said/VBD today/NN ./.
```

```
(ROOT
  (S
    (S
      (NP
        (NP (DT The) (JJS strongest) (NN rain))
        (VP
          (ADVP (RB ever))
          (VBN recorded)
          (PP (IN in)
            (NP (NNP India)))))
      (VP
        (VP (VBD shut)
          (PRT (RP down))
          (NP
            (NP (DT the) (JJ financial) (NN hub))
            (PP (IN of)
              (NP (NNP Mumbai)))))
        (, ,)
```

Ausgabe-Datei – Teil 2

```
      (VP (VBD snapped)
        (NP (NN communication) (NNS lines)))
      (, ,)
      (VP (VBD closed)
        (NP (NNS airports)))
      (CC and)
      (VP (VBD forced)
        (NP
          (NP (NNS thousands))
          (PP (IN of)
            (NP (NNS people))))
        (S
          (VP (TO to)
            (VP
              (VP (VB sleep)
                (PP (IN in)
                  (NP (PRP$ their) (NNS offices))))
              (CC or)
              (VP (VB walk)
                (NP (NN home))
                (PP (IN during)
                  (NP (DT the) (NN night)))))))))
  (, ,)
  (NP (NNS officials))
  (VP (VBD said)
    (NP (NN today)))
  (. .)))
```

Ausgabe-Datei – Teil 3 I

```
det(rain-3, The-1)
amod(rain-3, strongest-2)
nsubj(shut-8, rain-3)
nsubj(snapped-16, rain-3)
nsubj(closed-20, rain-3)
nsubj(forced-23, rain-3)
advmod(recorded-5, ever-4)
acl(rain-3, recorded-5)
case(India-7, in-6)
nmod:in(recorded-5, India-7)
ccomp(said-40, shut-8)
compound:prt(shut-8, down-9)
det(hub-12, the-10)
amod(hub-12, financial-11)
dobj(shut-8, hub-12)
case(Mumbai-14, of-13)
nmod:of(hub-12, Mumbai-14)
conj:and(shut-8, snapped-16)
ccomp(said-40, snapped-16)
compound(lines-18, communication-17)
```

Ausgabe-Datei – Teil 3 II

```
dobj(snapped-16, lines-18)
conj:and(shut-8, closed-20)
ccomp(said-40, closed-20)
dobj(closed-20, airports-21)
cc(shut-8, and-22)
conj:and(shut-8, forced-23)
ccomp(said-40, forced-23)
dobj(forced-23, thousands-24)
nsubj(sleep-28, thousands-24)
nsubj(walk-33, thousands-24)
case(people-26, of-25)
nmod:of(thousands-24, people-26)
mark(sleep-28, to-27)
xcomp(forced-23, sleep-28)
case(offices-31, in-29)
nmod:poss(offices-31, their-30)
nmod:in(sleep-28, offices-31)
cc(sleep-28, or-32)
xcomp(forced-23, walk-33)
conj:or(sleep-28, walk-33)
dobj(walk-33, home-34)
case(night-37, during-35)
det(night-37, the-36)
```

Ausgabe-Datei – Teil 3 III

```
nmod:during(walk-33, night-37)
nsubj(said-40, officials-39)
root(ROOT-0, said-40)
nmod:tmod(said-40, today-41)
```

Notationen im Stanford Parser

- Menge der POS-Tags und -Kategorien
  - Englisch – aus Penn Treebank
  - Chinesisch – aus Penn Chinese Treebank
  - Deutsch – aus NEGRA Korpus
- Menge der grammatischen Funktionen
  - Universal Dependecies
    `http://universaldependencies.org/`
  - ältere, sprachspezifische Formalismen auch noch verfügbar

Wichtige Dependenzen I

- **`nsubj`**: nominal subject
  - Subjekt in einer Phrase
  - Hauptwort ist nicht unbedingt ein Verb (z.B. *The flower is blue* - **`nsubj(blue,flower)`**)
- **`nsubjpass`**: passive nominal subject
  - Subjekt in einer Phrase mit dem Verb im Passiv

Wichtige Dependenzen II

- `dobj`: direct object
  - Direktes Objekt in einer Phrase
  - Immer im Akkusativ
- `iobj`: indirect object
  - Indirektes Objekt in einer Phrase
  - Immer im Dativ

Wichtige Dependenzen III

- `advmod`: adverbial modifier
  - Adverbialer Modifikator
- `amod`: adjectival modifier
  - Adjektivaler Modifikator
- `nmod`: nominal modifier
  - Nominaler Modifikator

Parser-Ausgabe exportieren

- Man kann Konstituentenparsebäume in das CoNLL-Format exportieren
- Option: `-conllx`
- Mit `-keepPunct` wird die Interpunktion behalten
- Beispiele:

```
:~$ java edu.stanford.nlp.trees.UniversalEnglishGrammatical\
Structure -treeFile file.tree -collapsedTree -conllx -keepPunct
```

Weitere Stanford NLP Tools

Stanford bietet nicht nur den Parser

- POS Tagger
  `http://nlp.stanford.edu/software/tagger.shtml`
- Named Entity Recognizer
  `http://nlp.stanford.edu/software/CRF-NER.shtml`
- Word Segmenter
  `http://nlp.stanford.edu/software/segmenter.shtml`
- ...
- zusamengefasst in CoreNLP:
  `http://stanfordnlp.github.io/CoreNLP/`

Übung 10

Mittagspause

Mittwoch: Python für NLP

7 Python für NLP
- NLTK
- spaCy

Python für NLP

Was ist NLTK?

- Zusammenstellung von Python-Modulen für NLP-Lehre, -Forschung und -Entwicklung
- Code für Vielzahl von Aufgaben aus der Sprachverarbeitung
- 40 bekannte Korpora
- ausführliche Dokumentation inkl. Online-Buch

Ein einfaches Beispiel

- Einbindung in Python-Programme:

```
>>> text = 'NLTK is my favourite natural language toolkit.'
>>> import nltk
>>> nltk.WhitespaceTokenizer().tokenize(text)
['NLTK', 'is', 'my', 'favourite', 'natural', 'language',
 'toolkit.']
```

Was NLTK bietet

- Infrastruktur als Grundlage für NLP-Programme in Python:
  1. Grundlegende Klassen zur Repräsentation NLP-relevanter Daten
  2. Standard-Schnittstellen für übliche Aufgaben, z.B. Tokenisierung, Tagging, Parsing
  3. Demos, z.B. Parser, Chunker, Chatbots
  4. Ausführliche Dokumentation, Tutorien, Referenzen

Dokumentation

- Online-Buch: `http://www.nltk.org/book/`
- API-Dokumentation: `http://www.nltk.org/api/`

Hilfe

- Online-Hilfe-Funktion:

```
>>> help(nltk.stem.WordnetStemmer)
class WordnetStemmer(nltk.stem.api.StemmerI)
 |  A stemmer that uses Wordnet`s built-in morphy function.
 |
 |  Method resolution order:
 |      WordnetStemmer
 |      nltk.stem.api.StemmerI
...
```

- Doc-Strings:

```
>>> print nltk.tag.BrillTagger.__doc__

    Brill's transformational rule-based tagger.  Brill
    taggers use an X{initial tagger} (such as
    L{tag.DefaultTagger}) to assign an intial tag
    sequence to a text; and then apply an ordered list
    of ...
```

Module

- NLTK ist in *Module* organisiert, die unterschiedliche Funktionalität bereitstellen.
- Wir schauen uns einige Module beispielhaft genauer an.

Modul `grammar` I

- Klassen zur Repräsentation kontextfreier Grammatiken (CFGs), Regeln, Nichtterminalzeichen, probabilistischer CFGs
- Funktionen zum Prüfen der Abdeckung, Induzieren probabilistischer CFGs, Parsen von Grammatiken, ...

Modul grammar II

```
>>> from nltk import grammar
>>> gr = grammar.CFG.fromstring("""
... S -> NP VP
... PP -> P NP
... NP -> Det N | NP PP
... VP -> V NP | VP PP
... Det -> 'a' | 'the'
... N -> 'dog' | 'cat'
... V -> 'chased' | 'sat'
... P -> 'on' | 'in'
... """)
>>> gr
<Grammar with 14 productions>
>>> gr.check_coverage("a cat chased a mouse".split())
ValueError: Grammar does not cover some of the input
words: u"'mouse'".
```

http://www.nltk.org/howto/grammar.html

Modul `corpus`

- Korpus-Reader für 40 NLTK-interne & -externe Korpora
- Beispiele für Funktionen einzelner Reader: `word()`, `sents()`, `paras()`, `tagged_words()`, `chunked_sents()`, `parsed_paras()`, `raw()`

```
>>> from nltk.corpus import brown
>>> print brown.words()
['The', 'Fulton', 'County', 'Grand', 'Jury',
 'said', ...]
```

Modul tag

- Klassen für verschiedene Taggermodelle: AffixTagger, BrillTagger, NGramTagger, HMM-Tagger

```
>>> from nltk import corpus, tag
>>> tagger = tag.UnigramTagger(corpus.brown.tagged_sents(),
...                            backoff=tag.DefaultTagger('NN'))
>>> tagger.tag(['The', 'dog', 'chewed', 'on', 'a', 'bone', '.'])
[('The', 'AT'), ('dog', 'NN'), ('chewed', 'VBD'), ('on', 'IN'),
('a', 'AT'), ('bone', 'NN'), ('.', '.')]
```

Modul `probability`

- Klassen zur Berechnung von Statistiken über Ergebnisse von Experimenten, unter anderem
  - FreqDist: Häufigkeitsverteilung
  - ConditionalFreqDist: Bedingte Häufigkeitsverteilungen, z.B.: Wie oft wird ein Wort mit welchem Tag versehen?

```
>>> from nltk import probability, tokenize
>>> probability.FreqDist(
...     tokenize.word_tokenize("this is a sentence"))
FreqDist({'this': 1, 'a': 1, 'is': 1, 'sentence': 1})
```

Beispiel: Satzgenerierung I

(Beispiele entnommen aus Madnani (2007): `http://www.umiacs.umd.edu/~nmadnani/pdf/crossroads.pdf`)

```
>>> from nltk.corpus import gutenberg
>>> from nltk.probability import ConditionalFreqDist
>>> from random import choice

# Create distribution object
>>> cfd = ConditionalFreqDist()

# For each token, count current word given previous word
>>> prev_word = None
>>> for word in gutenberg.words('austen-persuasion.txt'):
...     cfd[prev_word][word] += 1
...     prev_word = word
```

Beispiel: Satzgenerierung II

```
# Start predicting at the given word, say 'therefore'
>>> word = 'therefore'
>>> i = 1

# Find all words that can possibly follow the current word
# and choose one at random
>>> while i < 20:
...     print word
...     lwords = []
...     for next_word, count in cfd[word].items():
...         lwords.extend([next_word]*count)
...     follower = choice(lwords)
...     word = follower
...     i += 1

therefore it known of women ought . Leave me so well
placed in five altogether well placed themselves delighted
```

Weitere Module

Weitere Module sind unter anderem:

- classify
- metrics
- parse
- sem
- sentiment

Übung 11

Python für NLP

7 Python für NLP
- NLTK
- spaCy

Was ist spaCy?

- Python-Module für NLP-Entwicklung
- Fokus auf Geschwindigkeit und State-of-the-Art-Performanz
- Konzentration auf wenige essentielle Aufgaben
- für jede Aufgabe nur ein Algorithmus implementiert
- unterstützte Sprachen: Englisch, Deutsch, Französisch, Spanisch

Ein Beispiel

```
>>> import spacy
>>> nlp = spacy.load("en")
>>> doc = nlp(u"I love computational linguistics!")
>>> for token in doc:
...     print token, token.tag_, token.prob

I PRP -4.064180850982666
love VBP -7.974212169647217
computational JJ -19.579313278198242
linguistics NNS -19.579313278198242
! . -5.515825748443603 5
```

NLTK vs spaCy

NLTK:

- gut geeignet für Lehre
- enthält viele Algorithmen und Schnittstellen
- gut dokumentiert

spaCy:

- Idee: “getting things done”
- schneller und bessere Ergebnisse als NLTK
- kann sehr gut im Pre-Processing verwendet werden (auch für Deutsch)!

Dokumentation

- API-Dokumentation, Beispiele und Tutorials:
  `https://spacy.io/docs/`

Daten in spaCy

```
>>> doc = nlp(u"I love computational linguistics!")
```

- Annotationen gespeichert in Objekt der `Doc`-Klasse
- einzelne Wörter: Objekte der `Token`-Klasse
- Zugriff auf Wörter/Sequenzen: `doc[i]`, `doc[(i, j)]`
- annotiert: Tokenisierung, Satz-Splitting, POS-Tags, Dependenzbäume, Named Entity Recognition, Sentiment
- weitere Informationen: Wahrscheinlichkeit nach Language Model, Wort-Vektoren

Informationen auf Wortebene

```
>>> doc = nlp(u"I love computational linguistics!")
>>> print doc[1].tag_
```

Alle Attribute: `https://spacy.io/docs/api/token`

Informationen auf Dokumentenebe

```
>>> doc = nlp(u"I love computational linguistics!")
>>> print doc.vector
```

Alle Attribute: `https://spacy.io/docs/api/doc`

Aufbau I

- man muss zunächst die Pipeline laden:

```
>>> import spacy
>>> nlp = spacy.load("en")
```

- die Pipeline erwartet Text(e) als Unicode-Strings:

```
>>> doc = nlp(u"""I love computational linguistics!
Green frogs are green""")
>>> docs = nlp.pipe(list_of_texts)
```

Aufbau II

- dann kann man über Dokumente, Sätze und Token iterieren:

```
>>> for token in doc:
...     print(token.lemma_)
```

```
>>> other_doc = nlp(u"I hate geometry.")
>>> for sentence in doc.sents:
...     print sentence, sentence.similarity(other_doc)
I love computational linguistics! 0.839319268367
Green frogs are green. 0.646078181155
```

Vorhandene Annotationen

- schon vorhandene Annotation zu nutzen ist auch möglich:

```
>>> import spacy
>>>
>>> nlp = spacy.load("en")
>>>
>>> tokens = [u"Just", u"some", u"example"]
>>> doc = spacy.tokens.Doc(nlp.vocab, words=tokens)
>>> nlp.tagger(doc)
>>> for tok in doc:
...     print tok.pos_, tok.tag_
```

https://spacy.io/docs/usage/processing-text

Übung 12