TreeTagger

Deborah Watty

POS-Tagging

Das ist ein Haus.

Artikel Verb Artikel Nomen

The 1977 PCs could only store two pages of data.

Modalverb Adverb

Wir wissen: “store” kann Nomen oder Verb sein. Unser Modell muss wissen, dass auf ein Hilfsverb und Adjektiv eher ein Verb als ein Nomen folgt

➔ Kombination aus Lexikon und Modell

“store”

“Laden”

Hidden Markov Models

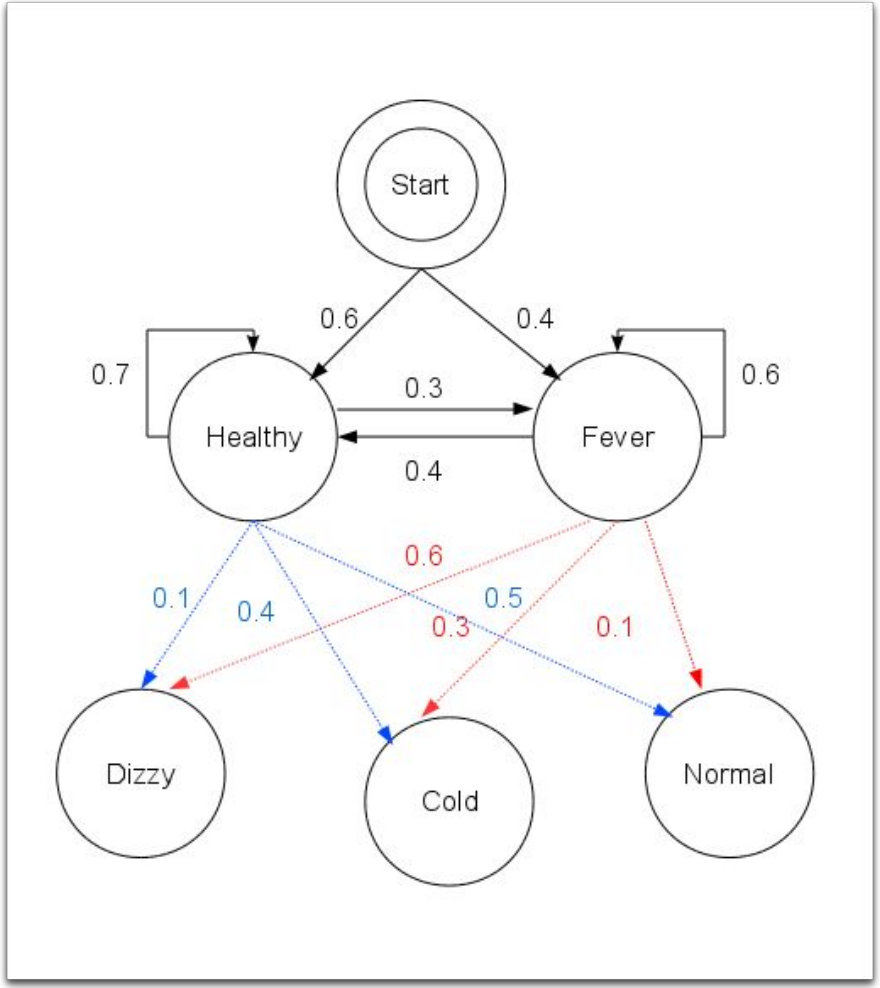
Idee: Beobachtete Ereignisfolge nutzen, um die Wahrscheinlichste Folge von Zuständen zu bestimmen

$$P(O,Q) = P(O|Q) \times P(Q) = \prod_{i=1}^{T} P(o_i|q_i) \times \prod_{i=1}^{T} P(q_i|q_{i-1})$$

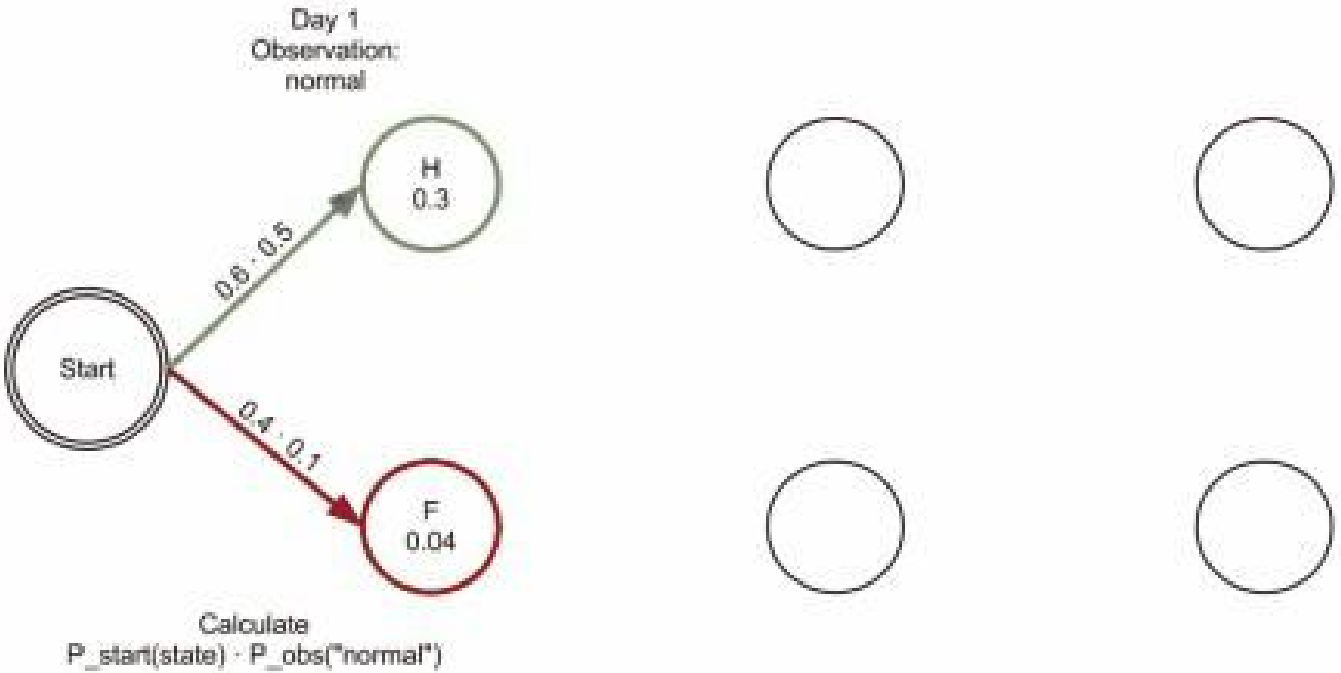
Beobachtung:

➔ Normal - Cold - Dizzy

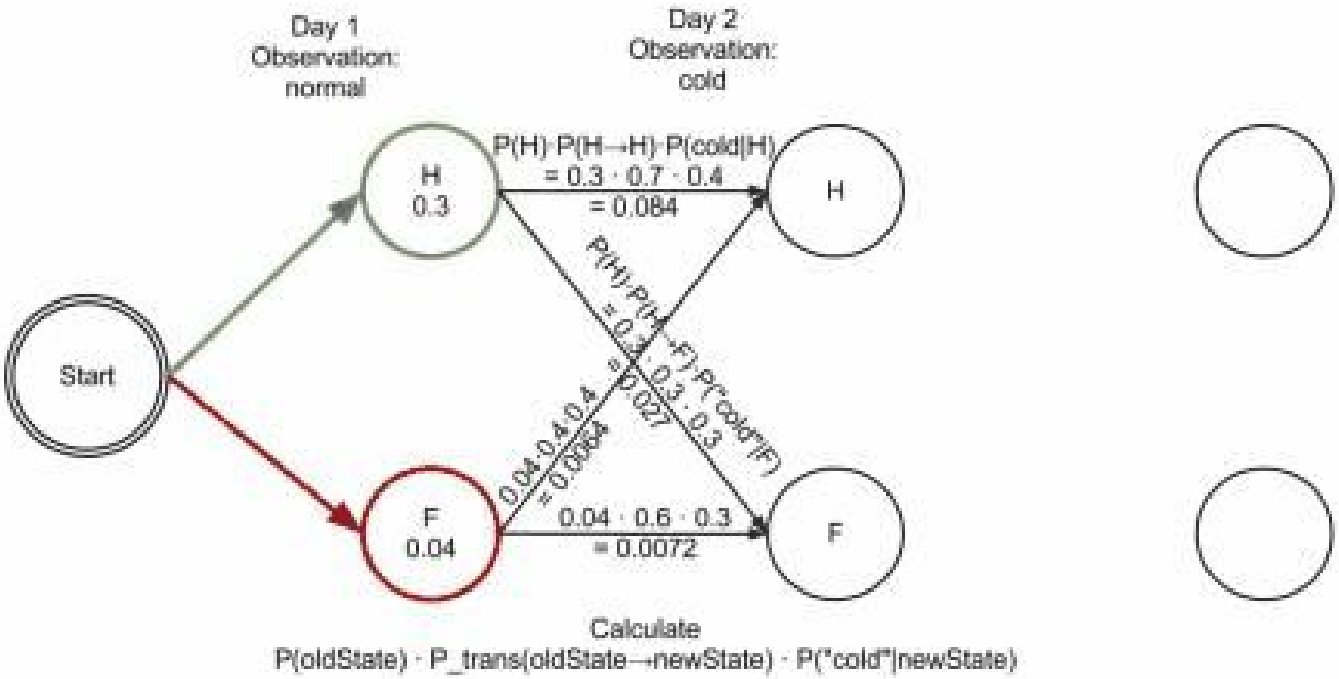
Woher bekommen wir die wahrscheinlichste Sequenz?



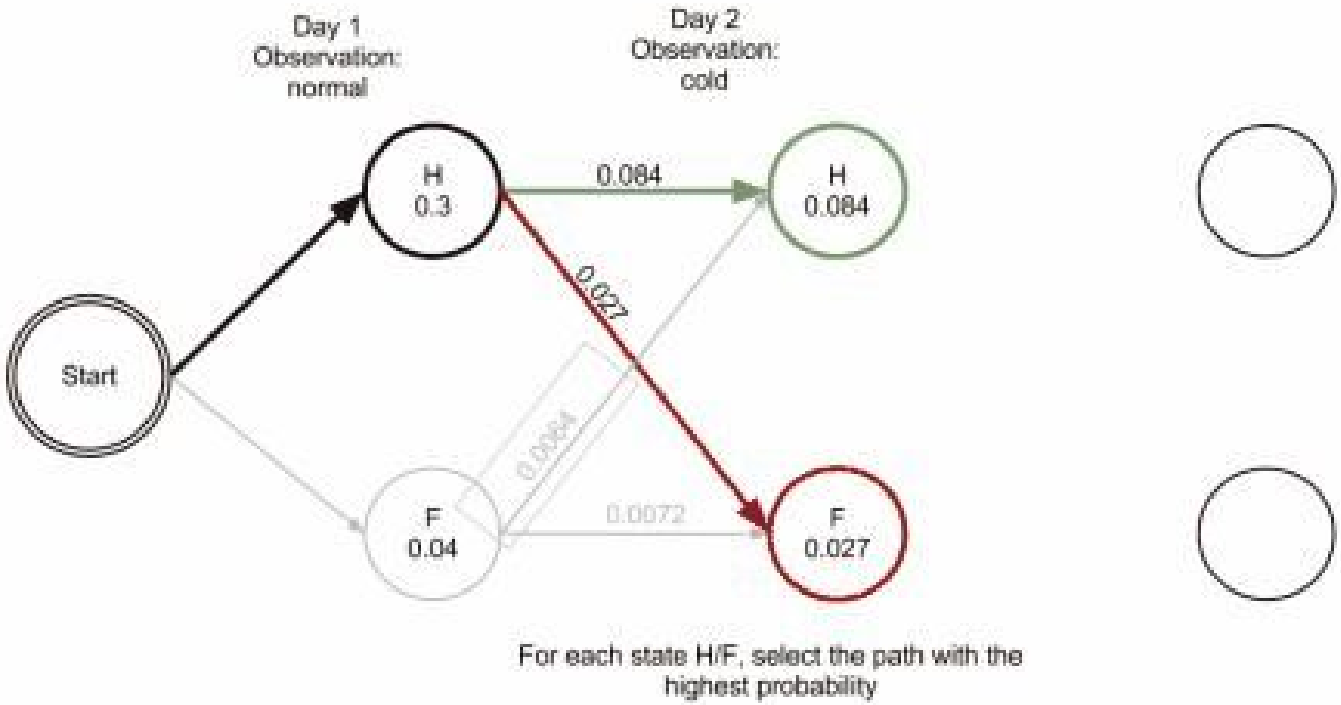
Viterbi-Algorithmus



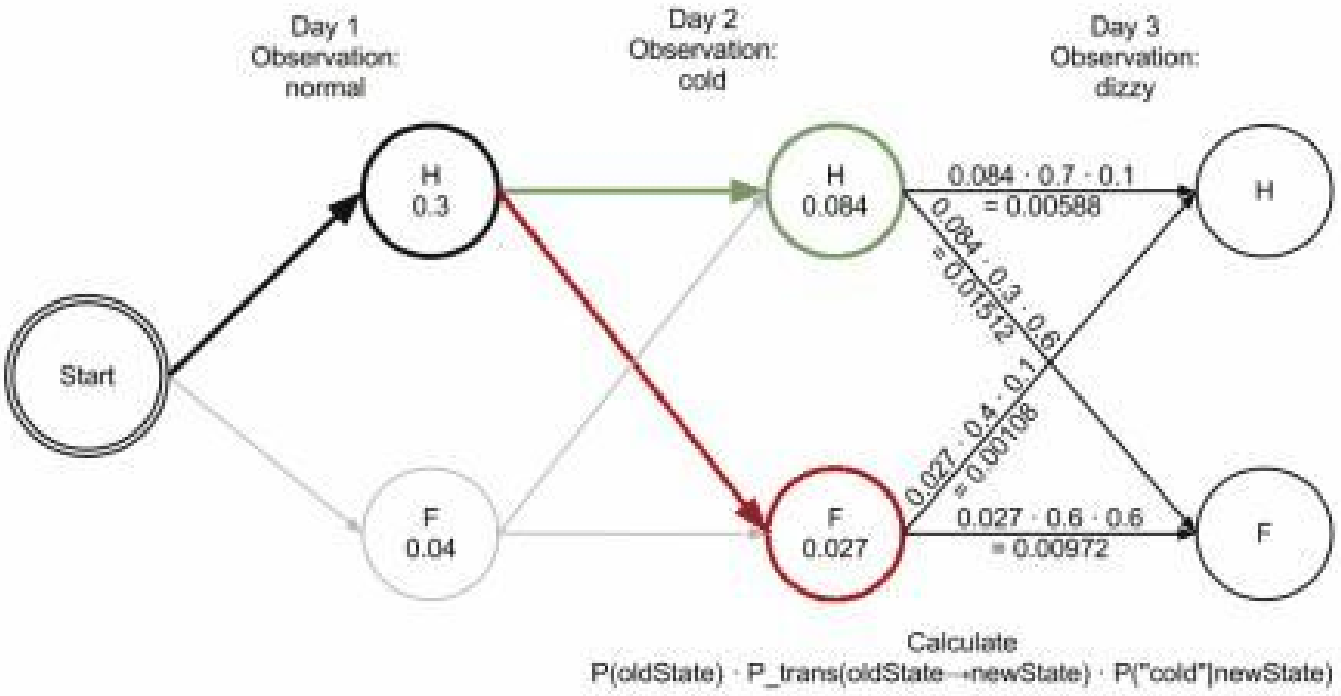
Viterbi-Algorithmus



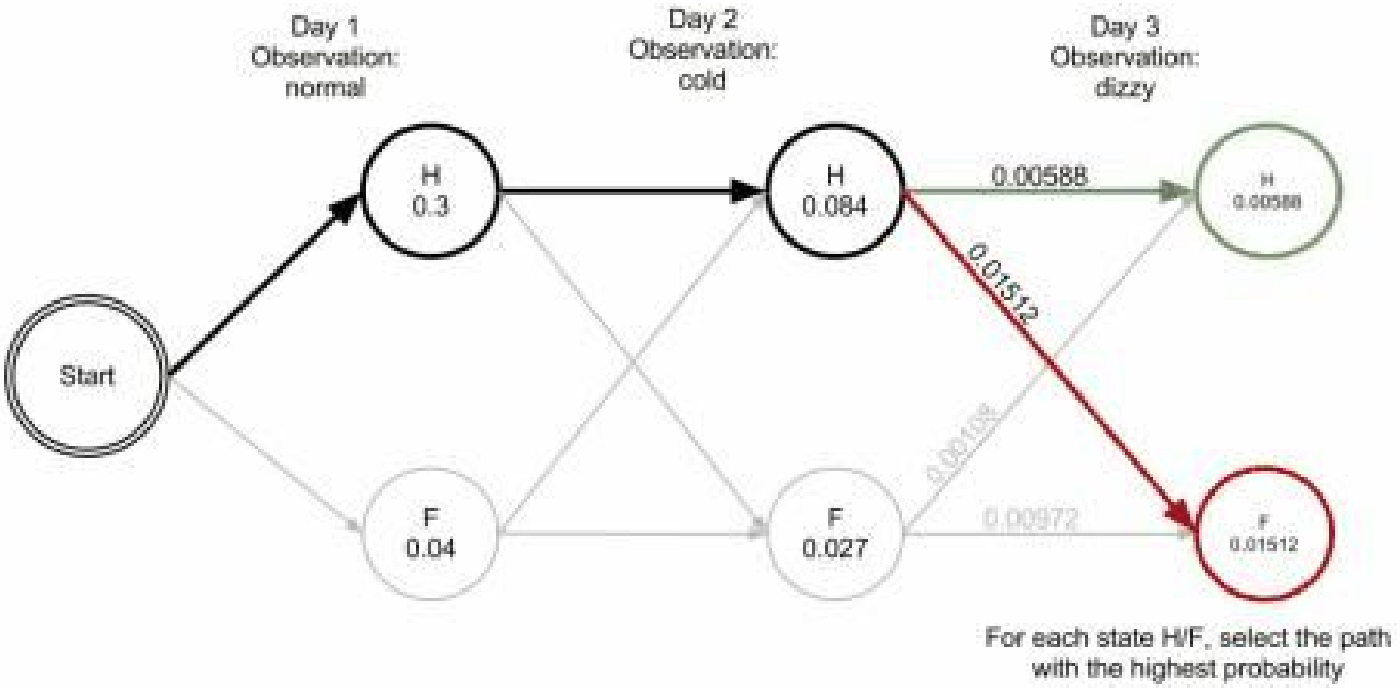
Viterbi-Algorithmus



Viterbi-Algorithmus



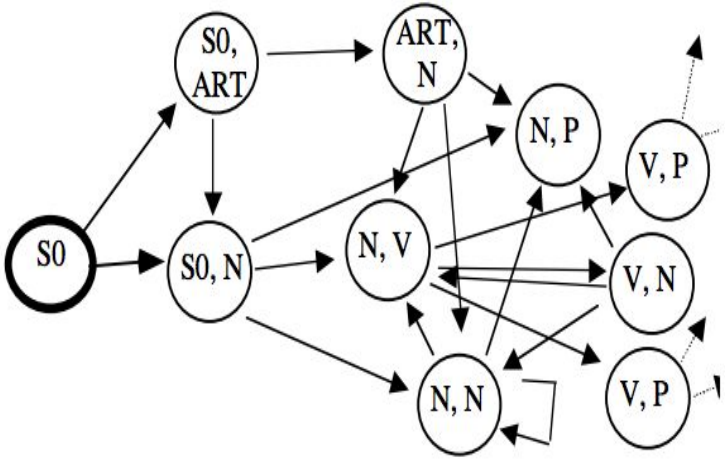
Viterbi-Algorithmus



Second Order HMMs

Beispiel:

- ➔ Tiger-Korpus mit 1 Million Tags
- ➔ STTS mit 54 Tags (plus Start und Ende)
- ➔ 54 Tags bedeuten ~1400 mögliche Zustände (Paar aus je zwei Wörtern)
- ➔ ~77000 verschiedene Übergänge
- ➔ Durchschnittlich 13 Trigramme pro Übergang
- ➔ Das sind unter Umständen zu wenig Daten



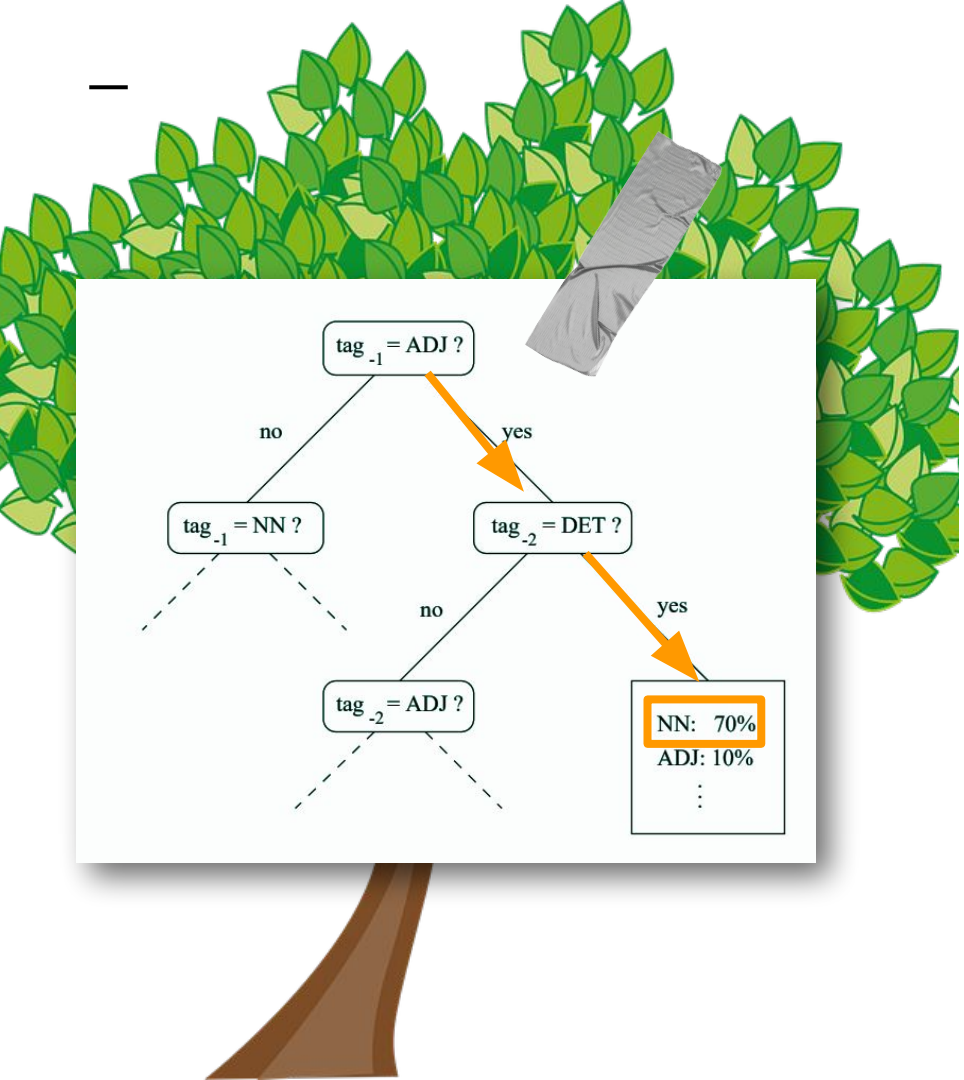
Credit: Arun Nedunchezhian

TreeTagger

- ➔ 1994 von Helmut Schmid (damals Uni Stuttgart) entwickelt
- ➔ Bis heute in Gebrauch, auch wenn es neue Ansätze gibt (neuronale Netze)
- ➔ Nutzt Entscheidungsbäume statt Hidden Markov Models

$$P(w_1 w_2 \dots w_n, t_1 t_2 \dots t_n) = \prod_{i=1}^{n} P(t_i|w_i)/P(t_i)\ P(t_i|t_{i-k} \dots t_{i-1})$$

Aus dem Lexikon *Entscheidungsbaum*

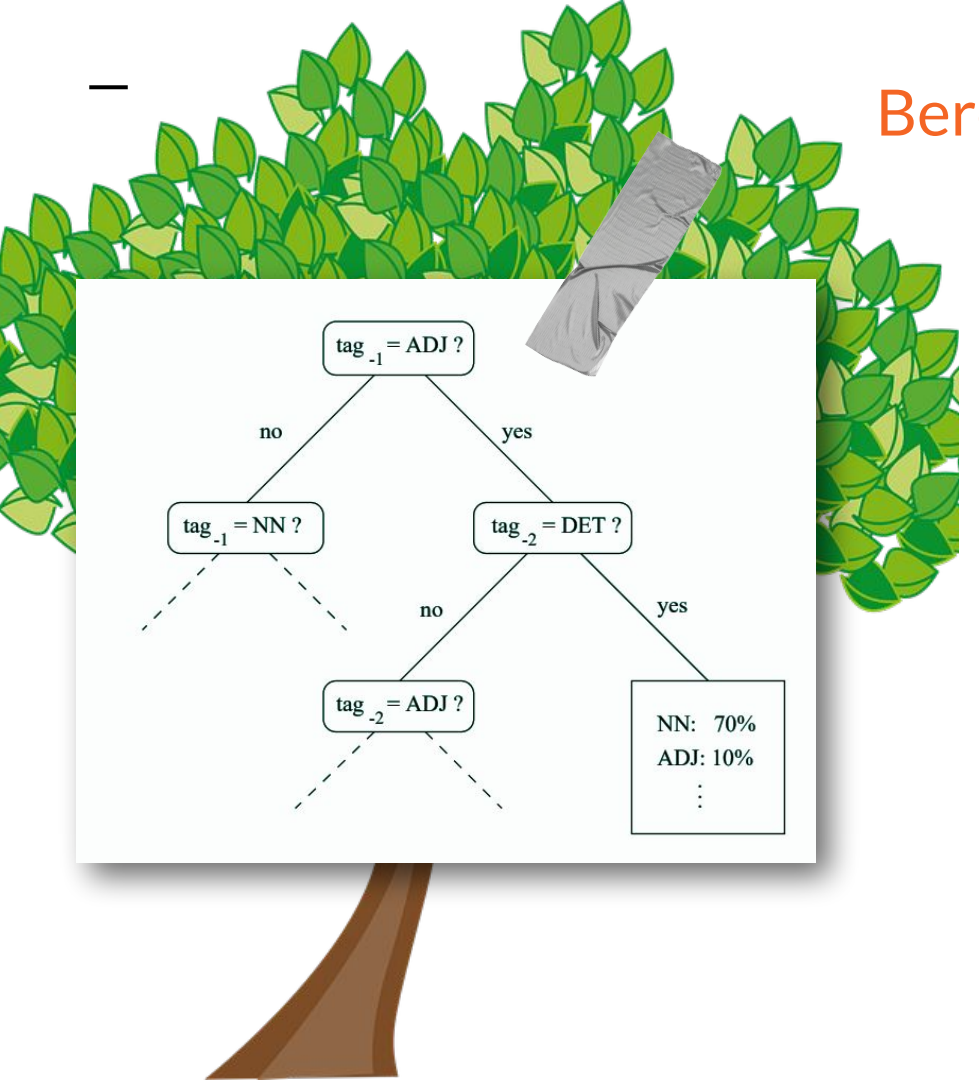


Decision Trees

Gesucht: P(NN|DET, ADJ) = 0,7

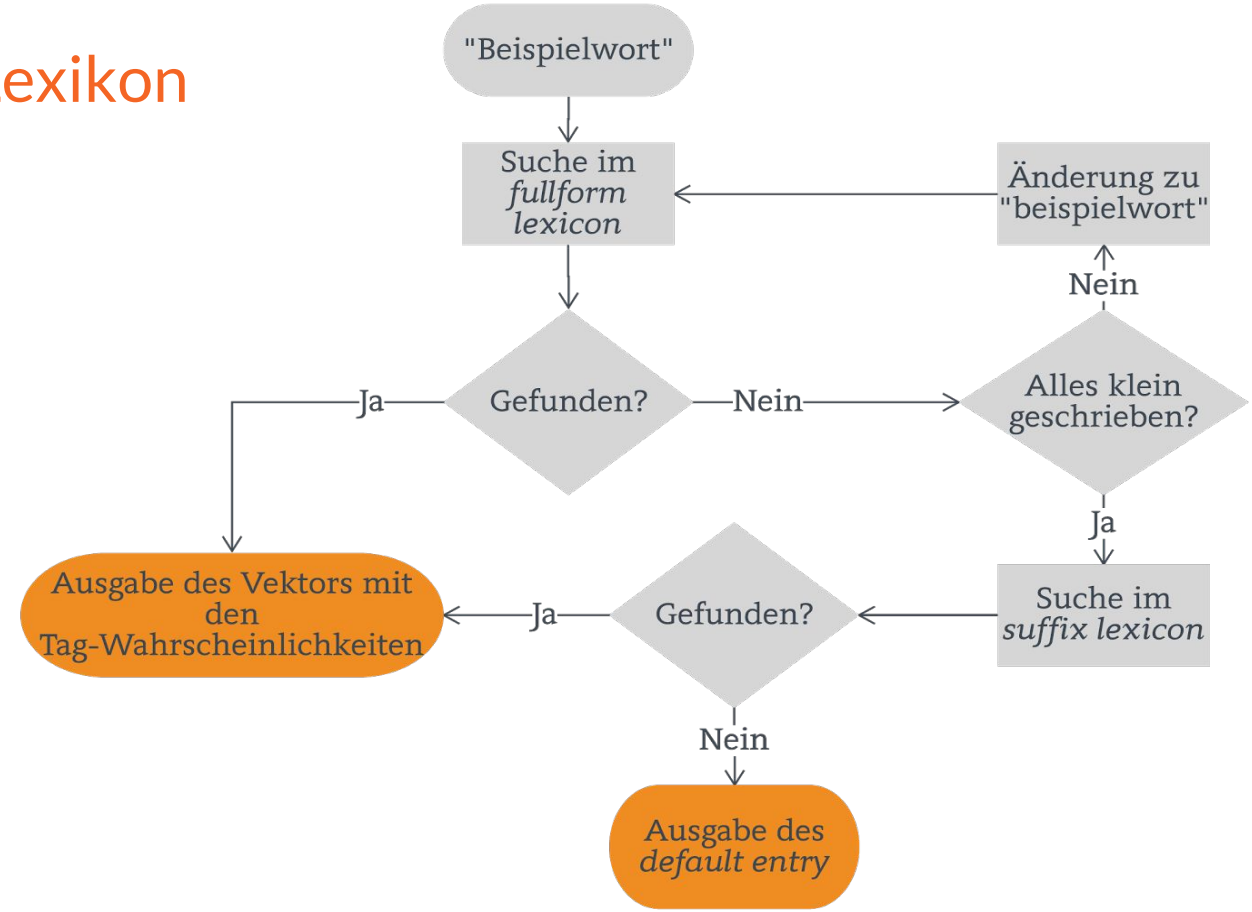
- ➔ Sobald zweimal ja gesagt wird, wird der Wahrscheinlichkeitsvektor ausgegeben
- ➔ Beispiel STTS: Maximal 106 Tests für 54 Tags
- ➔ Entscheidungsbäume werden durch einen Algorithmus berechnet, der die Tests so wählt, dass möglichst wenig Schritte gemacht werden müssen

Berechnung eines Decision Trees



- ➔ Alle Trigramme aus den Trainingsdaten werden aufgelistet
  [(DET ADJ NN), (ADJ NN DET), (NN DET V), (DET ADJ ADJ)]
- ➔ Der Test mit der größten Aussagekraft wird ausgewählt (“Information Gain”) und die Trigramme in zwei Listen aufgeteilt (Tags, die den Test bestehen und Tags, die den Test nicht bestehen
  [(DET ADJ NN)(DET ADJ ADJ)]
  [(ADJ NN DET), (NN DET V)]
- ➔ Für jede Liste wird Schritt 2 rekursiv durchgeführt
- ➔ Pruning: Wenn ein Knoten zwei Blätter als Kinder hat, die sich nicht wesentlich unterscheiden, werden die Kinder entfernt und der Knoten wird selbst ein Blatt

Suche im Lexikon

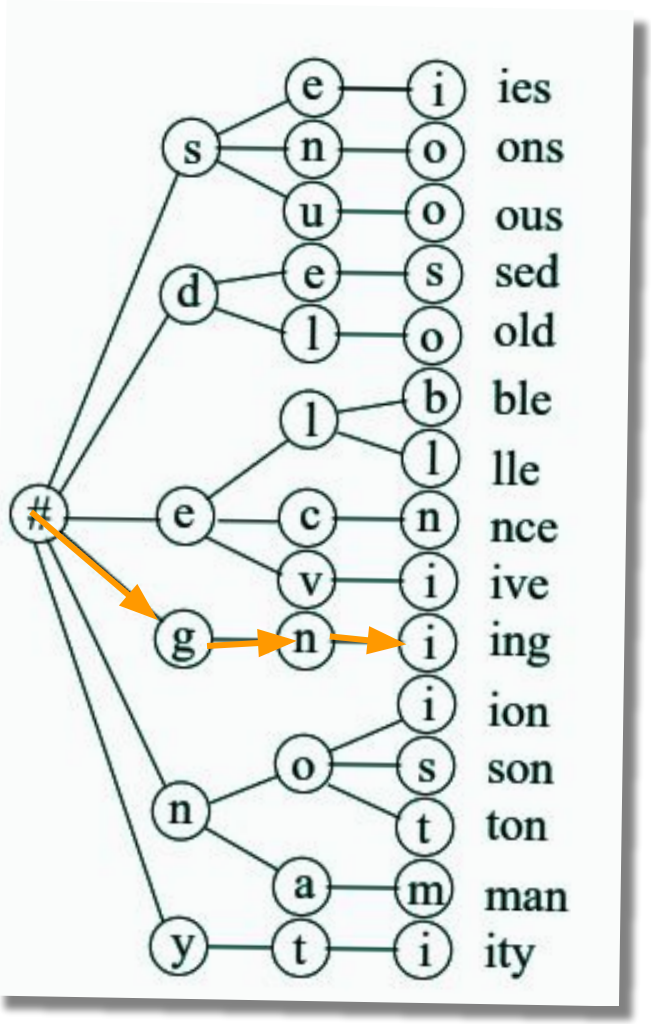


Suffix Lexicon

Gesucht: Wahrscheinlichkeitsvektor für “tagging”

Berechnung:

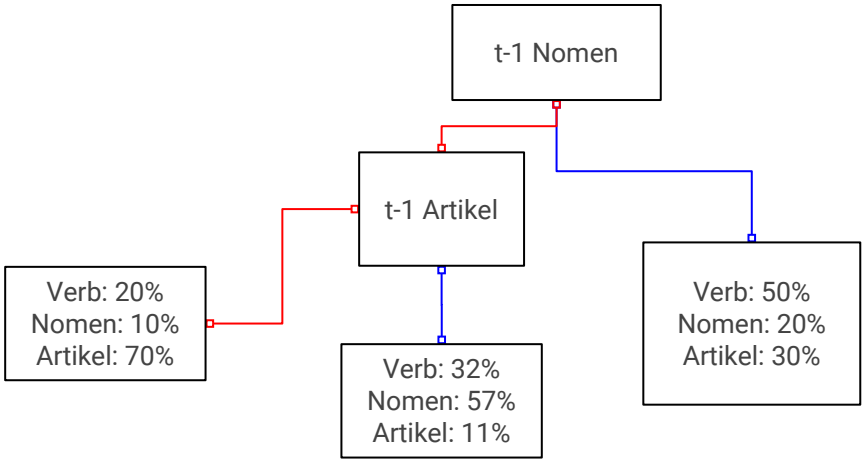
1. Liste aller Suffixe der Länge 5 [“bling”, “bring”, “sing”, “fries”, …]
2. Kürzen aller Suffixe, die nicht aussagekräftig genug sind
3. Berechnen der Wahrscheinlichkeiten P(Tag|Suffix)



Beispieldurchlauf

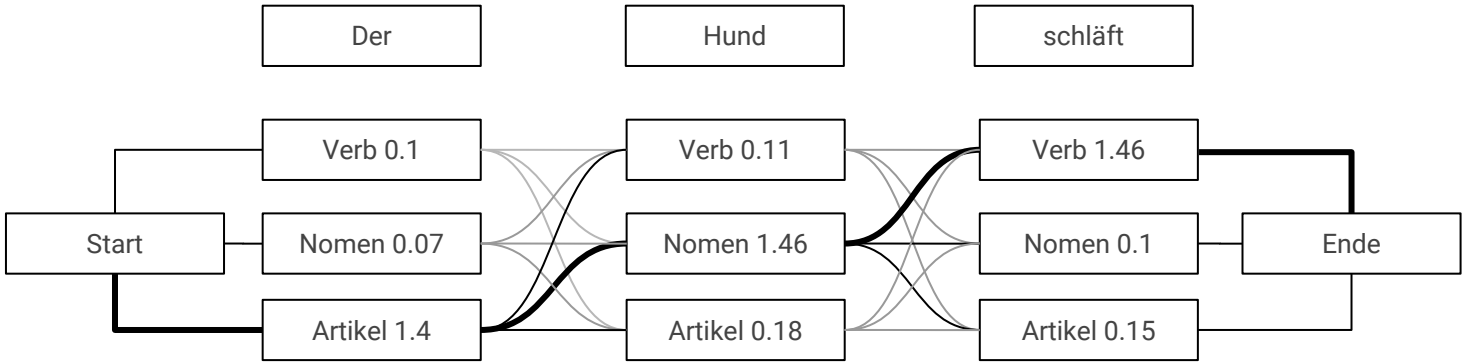
Der Hund schläft.

Tagset: Nomen (0.3), Artikel (0.3), Verb (0.4)



$$P(w_1 w_2 \dots w_n, t_1 t_2 \dots t_n) = \prod_{i=1}^{n} P(t_i|w_i)/P(t_i)\ P(t_i|t_{i-k} \dots t_{i-1})$$

Aus dem Lexikon *Entscheidungsbaum*



Lexikon:

Hund (0.1, 0.55, 0.35)

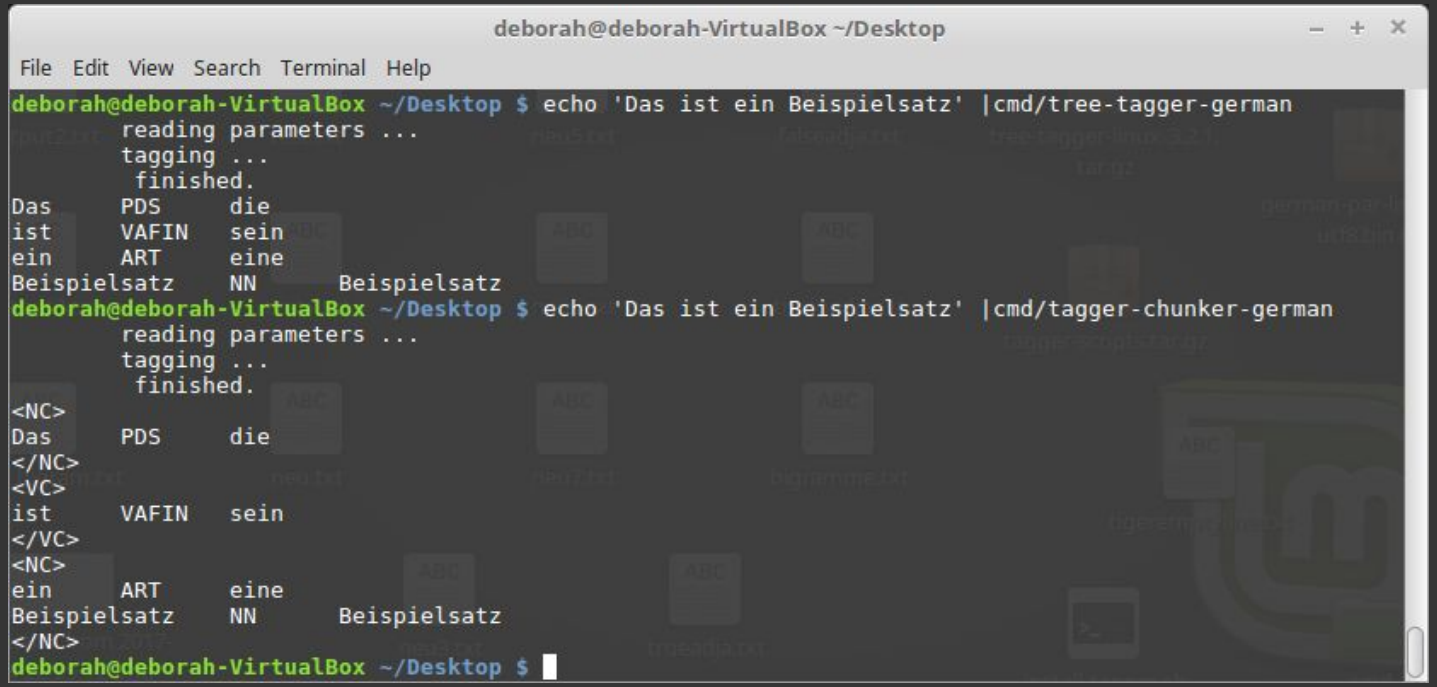
Der (0.2, 0.2, 0.6)

schläft (0.8, 0.1, 0.1)

spielt (0.6, 0.3, 0.1)

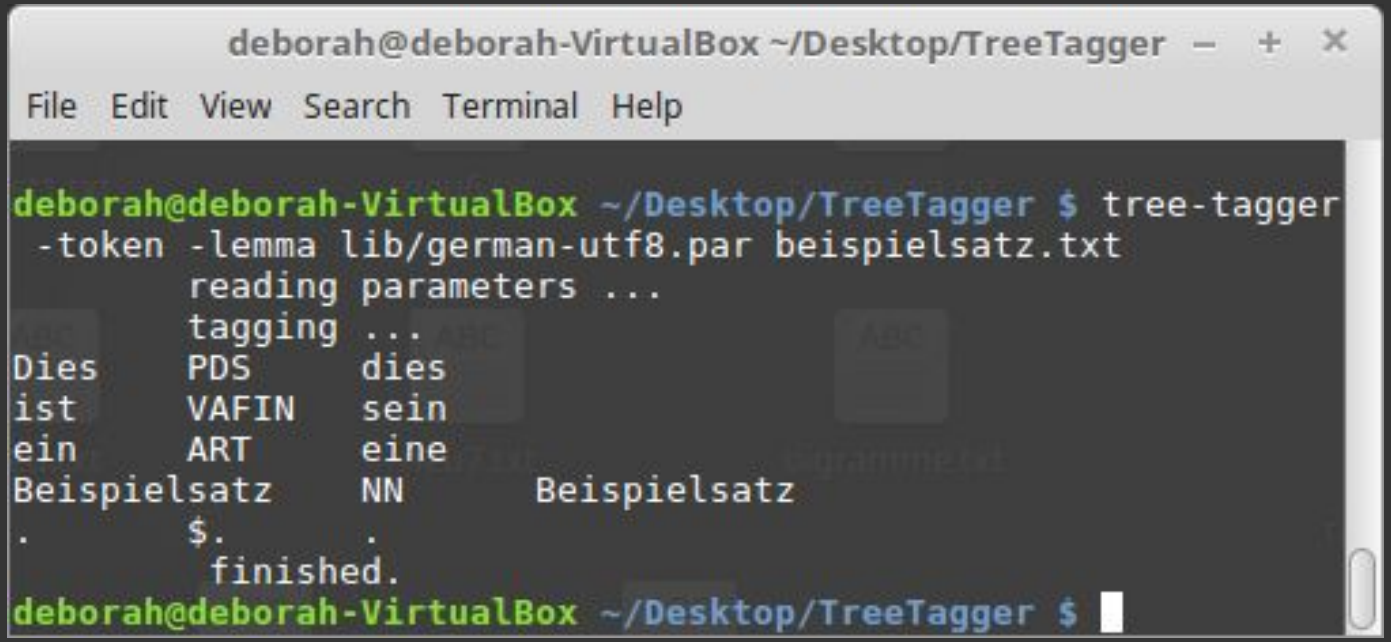
Installation testen

1. `echo 'Das ist ein Beispielsatz' |cmd/tree-tagger-german`
2. `echo 'Das ist ein Beispielsatz' |cmd/tagger-chunker-german`



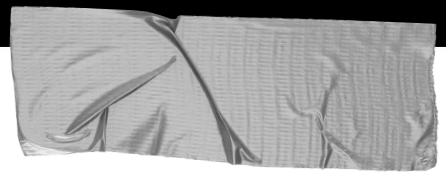
Übung: Einen einfachen Satz Taggen

```
tree-tagger -token -lemma lib/german-utf8.par beispielsatz.txt
```



Eine eigene Datei schreiben

1. vim
2. i
3. <Dateiinhalt schreiben>
4. Esc
5. :w <dateiname>
6. Enter
7. :q
8. (:q! falls Fehlermeldung)



```
He
moved
to
New York City  NP 1.0
and
bought
a
car
.
```

- ➔ Ein Wort/Satzzeichen pro Zeile
- ➔ Eventuell selbst vorher eine Liste von Tags hinter einem Wort hinzufügen

Übung: Eigenen Satz taggen

```
tree-tagger -token -lemma <parameter file> <input file> <output file>
```

Optionen

-token

-lemma

-prob

-threshold <p>

-cap-heuristics

-no-unknown

(weitere in der Dokumentation)

Parameter Files

lib/english-chunker-utf8.par

lib/english-utf8.par

lib/french-utf8.par

lib/german-chunker-utf8.par

lib/german-utf8.par

lib/spanish-utf8.par

Input

Schreibt eine kleine Input-Datei (wie auf der letzten Folie gezeigt) in einer Sprache, die ihr beherrscht und probiert verschiedene Optionen aus.

http://www.ims.uni-stuttgart.de/forschung/ressourcen/lexika/TagSets/stts-table.html (Liste mit Tags)
http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/ (Parameter Files für weitere Sprachen)

Wie kann man TreeTagger trainieren?

```
train-tree-tagger <lexicon> <open class file> <input file> <output file>
```

➔ <lexicon>:

aback RB aback

abacuses NNS abacus

abandon VB abandon VBP abandon

abandoned JJ abandoned VBD abandon VBN abandon

abandoning VBG abandon

(Wort \t Tag Lemma Tag Lemma \n)

TreeTagger wird schon für 23 Sprachen verwendet und kann ohne Probleme auf weitere Sprachen trainiert werden.

Wie kann man TreeTagger trainieren?

```
train-tree-tagger <lexicon> <open class file> <input file> <output file>
```

➔ <open class file>:

FW JJ JJR JJS NN NNS NP NPS RB RBR RBS VB VBD VBN VBP VBZ

(alle möglichen Tags)

TreeTagger wird schon für 23 Sprachen verwendet und kann ohne Probleme auf weitere Sprachen trainiert werden.

Wie kann man TreeTagger trainieren?

```
train-tree-tagger <lexicon> <open class file> <input file> <output file>
```

➔ <input file>:

Pierre NP

Vinken NP

, ,

61 CD

years NNS

(Korpus: Wort \t Tag \n)

TreeTagger wird schon für 23 Sprachen verwendet und kann ohne Probleme auf weitere Sprachen trainiert werden.

Wie kann man TreeTagger trainieren?

```
train-tree-tagger <lexicon> <open class file> <input file> <output file>
```

➔ <output file>:

Dateiname für neues Parameter File

TreeTagger wird schon für 23 Sprachen verwendet und kann ohne Probleme auf weitere Sprachen trainiert werden.

Universal POS Tags

- ➔ Nur 17 Tags, die auf alle Sprachen anwendbar sein sollen
- ➔ Das STTS hat 54 Tags - genauer, aber sehr spezifisch für die deutsche Sprache und schwieriger zu taggen
- ➔ Weitere Informationen: http://universaldependencies.org/u/pos/

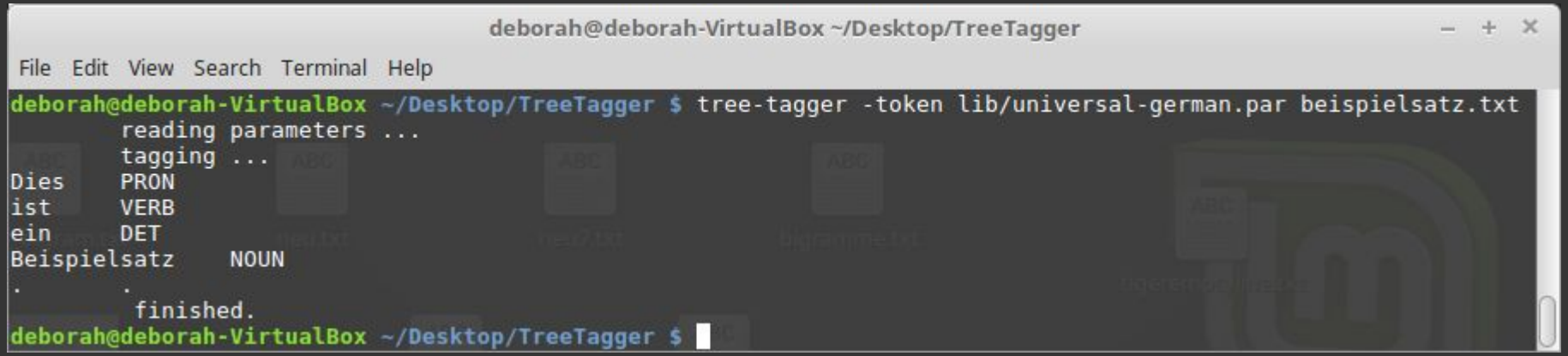
Alphabetical listing

- ADJ: adjective
- ADP: adposition
- ADV: adverb
- AUX: auxiliary
- CCONJ: coordinating conjunction
- DET: determiner
- INTJ: interjection
- NOUN: noun
- NUM: numeral
- PART: particle
- PRON: pronoun
- PROPN: proper noun
- PUNCT: punctuation
- SCONJ: subordinating conjunction
- SYM: symbol
- VERB: verb
- X: other

Übung: TreeTagger trainieren

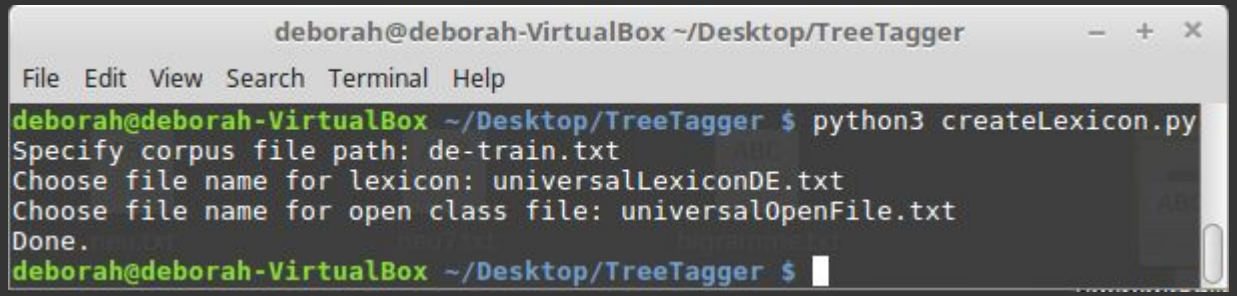
```
train-tree-tagger -st . universalLexiconDE.txt universalOpenFile.txt de-train.txt
lib/universal-german.par
```

`-st <p>` Tag für Satzzeichen falls nicht “SENT”



Übung: TreeTagger trainieren

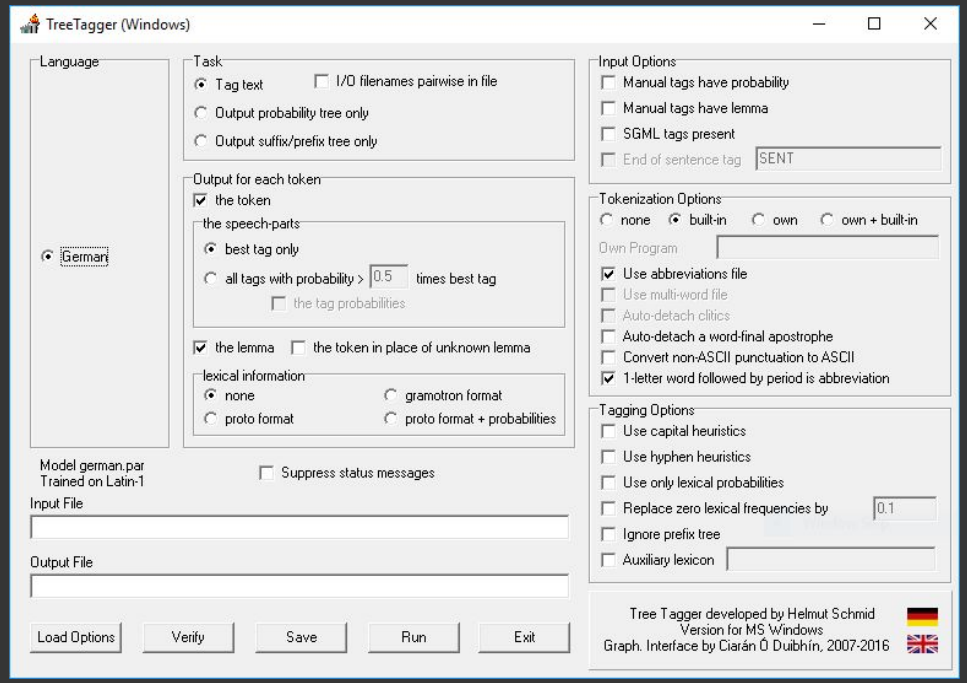
1. Eigene Korpus-Datei aussuchen (zum Beispiel Tiger-Korpus)
2. Mit createLexicon.py Lexikon und Open Class File erstellen

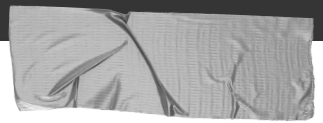


3. `train-tree-tagger <lexicon> <open class file> <input file> <output file>`

GUI

Leider nur für Windows verfügbar





Anleitung zur Installation

https://youtu.be/SYMc2Slll0c