

Satzgrenzenerkennung

Palmer & Hearst (1997)

Mikheev (2000) und (2002)

Jan Strunk

WS 2002/2003

Aktuelle Probleme der Computerlinguistik

Das SATZ System von Palmer & Hearst

Das SATZ System wird beschrieben in:

Palmer & Hearst (1997): Adaptive Multilingual Sentence Boundary Disambiguation. *Computational Linguistics Vol. 23 Issue 2.*

Palmer (1995): Experiments in multilingual sentence boundary recognition. *Proceedings of Recent Advances in Natural Language Processing, Bulgaria.*

Palmer & Hearst (1994): Adaptive Sentence Boundary Disambiguation. *Proceedings of the 4th ACL Conference for Applied Natural Language Processing, Stuttgart.*

Palmer (1994): SATZ – a adaptive sentence segmentation system. M.S. thesis. University of California at Berkeley.

SATZ: Zu lösendes Problem

Das SATZ System soll alle Satzgrenzen in einem Text identifizieren und markieren:

„The group included Dr. J.M. Freeman and T. Boone Pickens Jr.“

„Somit entsprach ein ECU am 17.9.1984 0.73016 US\$ (vgl. Tab. 1).“

SATZ: Vorgaben

- Korpus- und sprachunabhängig
 - Trainierbar
 - Verlässt sich nicht auf „brittle features“ wie Groß- und Kleinschreibung
- Auch für vollständig groß- oder kleingeschriebenen Text und OCR-Text geeignet.

SATZ: Grundidee

- Betrachten des lokalen Kontextes rund um einen Punkt
- Jedoch nicht der Token an sich, sondern ihrer Kategorien

Beispiel: *at the plant. He had thought*
 P Art N Prn V V

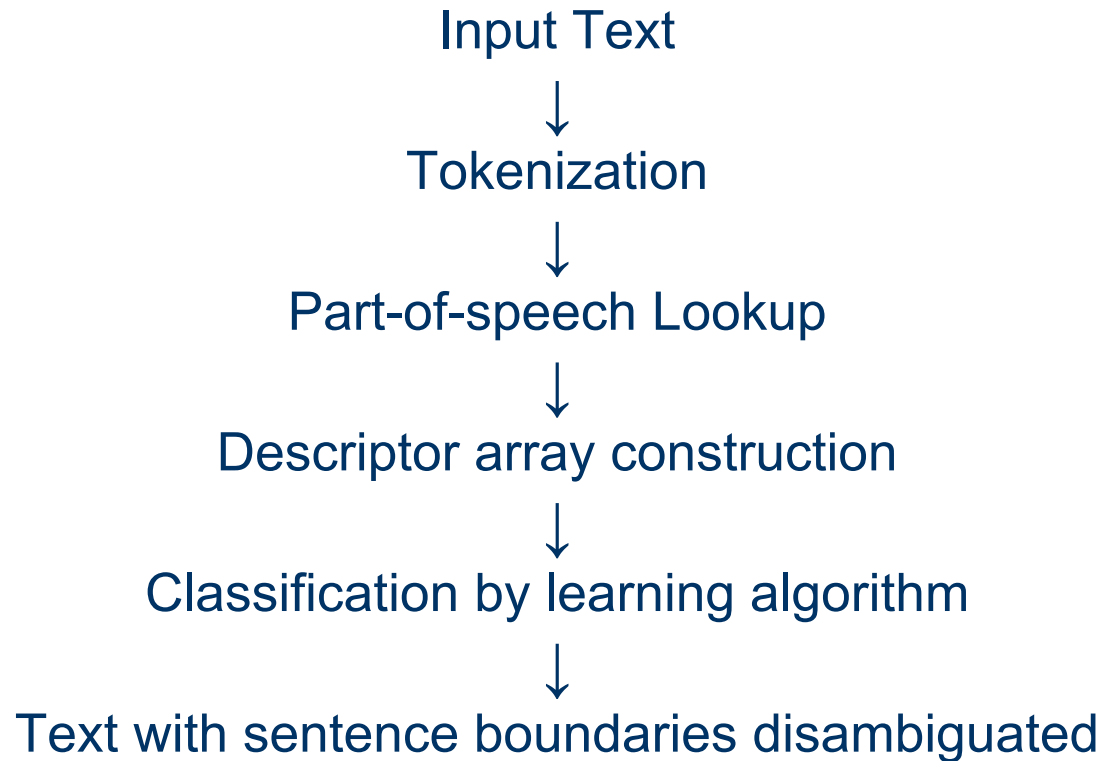
SATZ: Grundidee

- Problem: **Zirkularität**
 - Kategorien müssen erst durch Tagger ermittelt werden.
Tagger braucht Satzgrenzen.
- Lösung: Für jedes Token einen Kategorievektor

Beispiel: *at* *the* *plant* ← rel. Häufigkeit
 P(1.0) *Art(1.0)* *N(0.8)/V(0.2)*

He *had* *thought* ← binär
 Prn(1) *V(1)* *N(1)/V(1)*

SATZ: Verarbeitungsschema



SATZ: Part-of-speech Lookup

- Der User muss ein Lexikon aus einem getaggten Trainingskorpus oder aus Wortlisten erstellen.

Beispiel: *well* ***JJ/15 NN/18 QL/68 RB/634 UH/22
VB/5***

- Unbekannte Wörter werden mit Heuristiken kategorisiert.
- Wenn keine Kategorie gefunden werden kann, wird dem Token jeweils die gleiche Wahrscheinlichkeit für die Kategorien Nomen, Verb, Adjektiv und Abkürzung zugewiesen.

SATZ: Descriptor array construction

- Für jedes Token im Text wird ein Kategorievektor erstellt.
- Die Tags im Lexikon werden dabei auf nur 18 Kategorien abgebildet.
- Zusätzlich zwei Flags:
 - Großgeschrieben
 - Hinter Satzzeichen

SATZ: Descriptor array construction

- Probabilistisch

- Errechnung von Wahrscheinlichkeiten aus den relativen Häufigkeiten

$$\frac{\text{Token/Tag}}{\text{Token}}$$

- Binär

- Jede Kategorie, zu der das Token gehören kann, erhält den Wert 1 alle anderen den Wert 0.

SATZ: Classification by learning algorithm

- Kontextfenster von $k/2$ Token vor und $k/2$ Token hinter aktuell betrachtetem Token
- Ist das aktuell betrachtete Token ein mögliches Satzzeichen
 - Kategorievektoren des Kontextes an den Klassifikationsalgorithmus zum Klassifizieren oder Lernen (Training)

SATZ: Maschinenlernverfahren

- Grundsätzlich ist das SATZ-Verfahren unabhängig vom Lernalgorithmus
- Ausprobiert wurden:
 - Neuronale Netze mit unterschiedlicher Struktur
 - Eingabe: Kategorievektoren
 - Ausgabe: Wert zwischen 0 und 1
 - Klassifikation mittels oberem und unterem Schwellenwert
 - Training an Hand korrekter Eingabe-Ausgabe-Paare mit Back-propagation-Algorithmus

SATZ: Maschinenlernverfahren

– Decision Trees

- Quinlan c4.5 decision tree inductor
- Nur binäre Merkmale als Eingabe möglich
- Als Ausgabe ebenfalls nur binärer Wert möglich
- Baut einen Entscheidungsbaum nach den Eingabemerkmale auf
- Die Merkmale, mit dem höchsten Informationsgewinn, werden als erstes zur Entscheidung benutzt

SATZ: Beste Ergebnisse

Corpus	Size	Lower Bound	Baseline (STYLE)	SATZ NN	SATZ DT
WSJ	27,294	75.0 %	8.3 %	1.1 %	1.0 %
SDZ	3,184	79.1 %		1.3 %	1.9 %
German News	5,037	96.7 %		0.7 %	0.7 %
Hansards	3,766	80.1 %		0.6 %	0.4 %

SATZ: Zusätzliche Ergebnisse

- Beste Konfiguration:
 - Effektivster Kontext: -3 Token – +3 Token
 - Je größer das Lexikon, desto niedriger die Fehlerrate und desto schneller das Training.
 - Aber ein Lexikon mit 5000 Einträgen reicht aus.
 - Je mehr Trainingsdaten, desto besser die Ergebnisse.
 - Binäre Vektoren sind effektiver als probabilistische.
 - Der Entscheidungsbaum für Mixed-Case-Text benutzt nur 10 der 120 binären Attribute im Eingabevektor (der für Single-Case-Text sogar nur vier).

SATZ: Zusätzliche Ergebnisse

- Im Entscheidungsbaum benutzte Attribute
 - t-1 kann eine Abkürzung sein
 - t+1 ist ein Komma oder ein Semikolon
 - t+1 kann ein Satzendezeichen sein
 - t+1 kann ein Pronomen sein
 - t+1 ist groß geschrieben
 - t+1 kann eine Konjunktion sein
 - t+1 kann ein Eigenname sein
 - t+2 kann ein Nomen sein
 - t-3 kann ein Modifikator sein
 - t-3 kann ein Eigenname sein

SATZ: Zusätzliche Ergebnisse

- Gute Ergebnisse für Single-Case-Text
 - 3,3 % Fehler alles kleingeschrieben (WSJ)
 - 3,5 % Fehler alles großgeschrieben (WSJ)
- Relativ gute Ergebnisse für OCR-Text
 - 4,2 % Fehler an nicht bereinigtem Text (vs. 11,7 % Baseline)
 - 1,9 % Fehler an bereinigtem Text (vs. 9,6 % Baseline)

SATZ: Zusammenfassung

- Vorteile
 - Adaptation durch Training → sehr flexibel
 - Auch für Single-Case-Text und OCR-Text geeignet → sehr robust
 - Gute Fehlerraten
- Nachteile
 - Benötigte Ressourcen: Lexikon mit POS-Angaben, POS-Kategorien-Mapping, handgetaggte Trainingsdaten
 - Probleme: schlechte Ergebnisse ohne Abkürzungsliste (4,9 % Fehler auf dem WSJ), Homographen, verschachtelte Sätze

Die Systeme von Mikheev

Werden beschrieben in folgenden Artikeln:

Mikheev (2000): Tagging Sentence

Boundaries. In: *NACL'2000 Seattle*, S. 264-271.

Mikheev (2002): Periods, Capitalized Words,

etc. *Computational Linguistics Vol. 28 Issue 3*.

Mikheev (2000): Ziel und Vorgaben

- Das System soll alle Satzgrenzen innerhalb eines Korpus finden und markieren.
- Es soll dabei möglichst korpus- und sprachunabhängig sein.

Mikheev (2000): Grundidee

- Jeder Punkt wird grundsätzlich als eigenständiges Token betrachtet
 - Bsp.: *T . Boone Pickens Jr .*
- Als solches wird er mit Hilfe eines Taggers disambiguiert
 - Tagger berücksichtigt automatisch den lokalen Kontext
 - Es existiert eine Vielzahl von gut verstandenen Algorithmen für das Tagging-Problem
 - Ein HMM-Tagger berücksichtigt sowohl lexikalische Informationen als auch solche der Tag-Sequenz

Mikheev (2000): Grundidee

- Betrachten der syntaktischen Kategorien ermöglicht Erweiterung des verwendeten Kontextes
- Probleme des lexikalischen Ansatzes: Datenmangel, Homographe

Mikheev (2000): Umgehen des Zirkularitätsproblems

- Mikheev (2000) sieht keine Probleme beim Benutzen des Taggers, ohne vorher die Satzgrenzen zu kennen
- Grund für das Taggen in Sätzen ist rein programmtechnisch bedingt
- Lösung → Taggen einzelner Textabschnitte, an deren Ecken jeweils nicht ambige Sequenzen stehen

Mikheev (2000): Training und Tests

- Trainiert wurde der modifizierte Tagger im unsupervised mode auf dem WSJ und dem Brown Korpus
- Getestet wurde er auf dem jeweils anderen Korpus
- Ausgewertet wurden nur die Fehlerraten beim Erkennen von Satzendzeichen und beim Erkennen von Eigennamen bei Wörtern in „Großschreib“-Positionen

Mikheev (2000): Ergebnisse

System	Error on Sentence Punctuation		Error on Words in Mandatory Pos.	
	Brown Corpus	WSJ Corpus	Brown Corpus	WSJ Corpus
Upper Bound	0.01%	0.13 %	-	-
POS Tagger	0.25 %	0.39 %	3.15 %	4.72 %
POS Tagger Enhanced	0.20 %	0.31 %	1.87 %	3.22 %
POS Tagger / No Abbr. List	0.98 %	1.95 %	3.19 %	5.29 %
POS Tagger Enhanced / No Abbr. List	0.65 %	1.39 %	1.91 %	3.28 %

Mikheev (2000): Zusätzliche Ergebnisse

- Bei Single-Case-Text taggt der Tagger alle Abk. als Nicht-Satzenden (1,98 % Fehler beim WSJ)
- Abk. mit Eigennamen dahinter klassifiziert der Tagger immer als Nicht-Satzende
- Abkürzungsliste hat großen Einfluss auf Performanz
- Verbesserung des reinen Taggers durch Methoden zum Finden von Abkürzungs- und Eigennamen

Mikheev (2002): Ziele und Vorgaben

- Lösung wichtiger Probleme der Textnormalisierung:
 - Finden und Markieren der Satzgrenzen
 - Finden von Abkürzungen
 - Disambiguierung von großgeschriebenen Wörtern an Stellen, an denen immer großgeschrieben wird (Satzanfang, Überschriften).
- Flexibles, robustes, automatisch an neue Korpora adaptierbares System

Mikheev (2002): Grundideen

- Grundlegender Klassifikationsalgorithmus
 - Punkt nach Nichtabkürzung → Satzende
 - Punkt nach Abkürzung und am Absatzende → gleichzeitig Abkürzungs- und Satzpunkt
 - Punkt nach Abkürzung und folgendes nicht großgeschriebenes Wort → Abkürzungspunkt
 - Punkt nach Abkürzung und vor großgeschriebenem Wort, das kein Eigenname ist → Abkürzungs- und Satzpunkt

Mikheev (2002): Grundideen

- Dieser Algorithmus reicht nicht ganz aus
- Würde aber bei idealen Bedingungen nur 0.01 % Fehlerrate für das Brown Korpus und 0.13 % Fehlerrate für das WSJ Korpus bringen (Obergrenze)
- Vorher müssen jedoch die beiden Unterprobleme: Abkürzungs- und Eigennamenerkennung gelöst werden
- Baseline: Lexicon Lookup für Eigennamen und Abkürzungserkennungsheuristiken → 2.0 % Fehlerrate Brown Corpus; 4.1 % WSJ

Mikheev (2002): Grundideen

- Finden von Abkürzungen und Eigennamen mit dem Document Centred Approach (DCA)
 - Disambiguierung ambiger Token anhand ihres Vorkommens in nicht-ambigen Kontexten
 - Basiert auf der Verteilung eines Types im gesamten Dokument
 - Ist eine Art unsupervised learning on the fly = automatische Korpusfiltermethode

Mikheev (2002): Abkürzungserkennung

- Heuristiken
 - Kurzes Token ohne Vokale, nicht nur Großbuchstaben, Bsp.: *Mr., Dr., kg.*
 - Folge von Einzelbuchstaben und Punkten
Bsp.: *Y.M.C.A., e.g., i.e.*
 - Einzelner Buchstabe mit Punkt am Ende
- Automatisch aus großem Korpus extrahierte Abkürzungsliste (list lookup)

Mikheev (2002): Abkürzungserkennung

- DCA (Positional Guessing Strategy)
 - Voraussetzung: Konsistenz innerhalb eines Dokumentes
 - Positive Evidenz: Type mit Punkt am Ende kommt auch vor Kleinbuchstabe, Ziffer oder Komma vor
 - Negative Evidenz: Type mit Punkt kommt auch ohne Punkt im Text vor
 - Problem: Homographen, Bsp.: *in / in., no / no.*
 - Lösung: Extrahieren von Bigrammen mit vorhergehendem Wort zur Disambiguierung möglicher Homographen
Bsp.: *Vitamin C. Research* vs. *John C. Research*

Mikheev (2002): Abkürzungserkennung

- Beispiel

Abkürzungskandidat

*He lives in Hong **Kong**. That's a part of China.*

***Kong.**,*

*Hong **Kong** is great.*

Abkürzung

Nichtabkürzung

Mikheev (2002): Abkürzungserkennung

- Fehlerraten bei der Abkürzungserkennung

Methode	WSJ	Brown
Guessing Heuristics (GH)	9.6 %	10.8 %
List Lookup (LL)	12.6 %	11.9 %
GH + LL	1.2 %	2.1 %
GH + DCA	6.6 %	8.9 %
GH + DCA + LL	0.8 %	1.2 %

Mikheev (2002): Finden von Eigennamen

- Benutzung einer Kaskade von simplen DCA-Techniken
 - Sequence Strategy
 - Frequent List Lookup Strategy
 - Single Word Assignment
 - Heuristics
 - Lexicon Lookup Strategy (Nachschlagen in einer Liste „normaler“ Wörter aus einem anderen Korpus)

Mikheev (2002): Sequence Strategy

- Extraktion einer Folge von großgeschriebenen Wörtern aus nicht ambiger Position
 - Bsp.: ***Rocket System Co.***
- Generierung von Teilketten
 - Bsp.: ***Rocket System*** und ***System Co.***
 - Wenn so ein Kette oder Teilkette in ambiger Position gefunden wird, wird sie ebenfalls als Kette von Eigennamen betrachtet.
- Ebenfalls Extraktion kleingeschriebener Bigramme zum Finden von Nichteigennamen

Mikheev (2002): Frequent List Lookup

- Liste von Wörtern, die häufig am Satzbeginn stehen
- Liste von häufigen Eigennamen, die homograph mit normalen Wörtern sind
- Behandlung ambiger Token, die noch nicht von der Sequence Strategy klassifiziert worden sind

Mikheev (2002): Single Word Assignment

- Für jedes Wort in ambiger Position, das noch nicht disambiguiert worden ist:
 - Kommt es in nicht ambigen Positionen nur kleingeschrieben vor → kein Eigennamen
 - Kommt es in nicht ambigen Positionen großgeschrieben vor → Eigennamen
 - Probleme mit mangelnder Konsistenz
 - Bsp.: *the sheriff* vs. *Sheriff Jones*

Mikheev (2002): Heuristiken

- Großgeschriebenes Wort in Klammer → Eigenname; Bsp.: *John (Cool) Lee*
- Großgeschriebenes Wort nach kleingeschriebenem Wort, Komma oder Nummer und linker Klammer → Eigenname
- Großgeschriebenes Token nach großgeschriebener Abk. → Eigenname
 - Bsp.: *Mr. Smith*

Mikheev (2002): Evaluation für die Erkennung von Eigennamen

Methode	Fehlerrate		Abdeckung	
	WSJ	Brown	WSJ	Brown
Seq.	0.12 % 0.28 %	0.97 % 0.21 %	12.6 % 17.68 %	8.82 % 26.5 %
Freq. List	0.49 % 0.21 %	0.16 % 0.14 %	2.62 % 64.62 %	6.54 % 61.20 %
Single w.	3.18 % 6.51 %	1.96 % 2.87 %	18.77 % 3.07 %	34.14 % 4.78 %
Heuristics	< 2 %	< 2 %	?	?
Casc. DCA	1.10 %	0.76 %	84.12 %	91.76 %
Casc. + LL	4.88 %	2.83 %	100 %	100 %

Eigennamen

Normale Wörter

Mikheev (2002): Evaluation

	Brown Corpus			WSJ Corpus		
	SBD	Cap. W.	Abbrs.	SBD	Cap. W.	Abbrs.
UB	0.01 %	0.00 %	0.00 %	0.13 %	0.00 %	0.00 %
LB	2.00 %	7.40 %	10.8 %	4.10 %	15.0 %	9.6 %
DCA	0.28 %	2.83 %	0.8 %	0.45 %	4.88 %	1.2 %
DCA (no abbr. lex)	0.65 %	2.89 %	7.7 %	1.41 %	4.92 %	6.4 %
Tagger	0.25 %	3.15 %	1.2 %	0.39 %	4.72 %	2.1 %
Tagger + DCA	0.20 %	1.87 %	0.8 %	0.31 %	3.22 %	1.2 %

Mikheev (2002): Vorteile

- Bessere Ergebnisse als das SATZ-System
- Gleichzeitige Disambiguierung von großgeschriebenen Wörtern und Finden von Eigennamen
- Kein Training notwendig → passt sich automatisch jedem neuen Korpus an
- Auch für andere Sprachen geeignet (getestet am Russischen)

Mikheev (2002): Probleme

- Korpusfiltermethode → Probleme mit sehr kleinen Dokumenten (zu wenig Daten)
- Punkte nach Abk. und vor Eigennamen werden nicht disambiguiert
- Leistungsverlust ohne Abkürzungsliste
- Einige zusätzliche Wortlisten erforderlich, die aber automatisch generiert werden können
- Weniger für Single-Case- und OCR-Text geeignet
- Probleme für das Deutsche ← Großschreibung der Substantive

Fragen?

