Automatische Morphemanalyse

Johannes Stiehler

Wozu?

- noch immer viele europäische Sprachen ohne befriedigende Morphologie
- ermöglicht es, verschiedene Sprachstadien morphologisch zu vergleichen ohne Wochen mit der morphologischen Analyse von Altfranzösisch zuzubringen
- Ist Vorbereitung für „unsupervised grammar acquisition system“

Wie?

- funktioniert nur für Stamm+Suffix-Sprachen, da auf einem MDL-Teilungsalgorithmus basierend
- Performance: Pentium II 333MHz, 500000 Wörter ← 5 min.

Verschiedene Ansätze zur Automorphologie

Identifikation der Morphemgrenze

- Wie vorhersagbar ist n+1ste Buchstabe, wenn wir die ersten n Buchstaben gegeben haben?
- Auch interpretierbar als Weg von der Wurzel eines Tries zum Terminal.
- Lokale Spitzen zeigen Präfixe (von links) bzw. Suffixe (von rechts)
- Erweist sich als guter Algorithmus, um Morphemkandidaten zu finden (Precision 0.91, Recall 0.61) aber beinhaltet keine Möglichkeiten zur Verbesserung des Ergebnisses

Bigramme und Trigramme suchen, die mit hoher Wahrscheinlichkeit Morpheme sind

- Geht von einem Trainingskorpus aus
- Jedem Bigramm wird ein Tripel $< l, m, r >$ zugewiesen, dass die Wahrscheinlichkeit (vom Trainingscorpus aus gesehen) repräsentiert, dass links vom, im bzw. rechts vom Bigramm eine Morphemgrenze auftritt.
- Für das Wort „string“ errechnet sich die Wahrscheinlichkeit einer Morphemzerlegung str+ing durch l(„in“)+m(„ri“)+r(„tr“).
- Morphemgrenzen können entweder ab einem bestimmten Schwellwert oder bei örtlichen Spitzenwerten angenommen werden
- Zusatzannahmen: Wenn einer der drei Werte 0 ist, ist auch die Summe 0. Häufigkeit am Wortende wird vom Morphemwert ausgeschlossen lin+guist+ique, weil „lin“ oft am Wortende

Problem die meisten Bigramme tauchen genauso oft in Morphemen wie außerhalb von Morphemen auf → Morphologie operiert wohl auf einem höheren Level als der Buchstabeneben und hat nur geringe statistische phonologische Merkmale

Identifikation von Mustern

Das System wird mit gelabelten Wortpaaren gefüttert $< Grundform, Flexionsform >$ und soll aus dieser Information Grammatikregeln ableiten

Suche der „dichtesten“ Analyse

Der Algorithmus soll nach der Analyse suchen, die „das kleinste Lexikon produziert“, also z. B. Zahl der Buchstaben in der Wortliste vs. Zahl der Buchstaben in Stämmen und Suffixen des Analyseergebnisses.

DeMarcken(1995) - Algorithmus für Wortgrenzenerkennung

Wortgrenzenerkennung in Sprachen, die diese nicht orthografische kenntlich machen (Chinesisch) ist eine ähnliche Aufgabe

1. alle Buchstaben werden einzeln ins Lexikon eingetragen

2. nach und nach werden 2er, 3er etc. Verbände gebildet und überprüft, ob dieser Schritt zur Verkleinerung des Lexikons beiträgt. Verbände kommen dann ins Lexikon, wenn die Verkleinerung des Lexikons, die sie bewirken mehr ausmacht, als ihre eigene Länge.
3. die Länge eines Wortes ist $-\log F$ (Frequenz des Wortes)
4. die Größe des Gesamtsystems ist die Summe alle Wortlängen (s.o.) plus der des Lexikons
5. Verbände, die nur als Bestandteil eines einzelnen größeren Verbands vorkommen, werden eliminiert

Die Bewertung einer Stringanalyse ist die Summe von $-\log(F)$ der Substrings. Ein optimaler Parse erzielt einen möglichst kleinen Wert.

Ist auf Morphemanalyse (Splitten in zwei Teile) nicht direkt übertragbar, da das zu einem peripheren Schnitt führt. Außerdem wäre dann kein Unterschied zwischen häufigen Strings und echten Morphemen („schl“).

Goldsmiths Algorithmus

1. Nach bestimmten Heuristiken die erste Aufteilungsstelle finden.
2. Länge der Morphologie und des Korpus feststellen
3. Modifikation der Grammatik vornehmen
4. Länge der Morphologie und des Korpus mit der neuen Grammatik feststellen
5. Wenn kleiner als vorher, beibehalten, sonst alte beibehalten
6. Weiter mit (3)

Der Anfangs„split“

Strategie 1: Alle Wörter aufteilen

Notation: $|w|$ = Länge von w; $[w]$ = Häufigkeit im Korpus; W = Menge aller Wörter im Korpus; $[W]$ = Anzahl der Wörter im Korpus

1. Beim ersten Durchgang werden alle Parse-Möglichkeiten eines Wortes als gleich wahrscheinlich behandelt.

2. Jedem Stamm wird eine Häufigkeit von $\frac{[w]}{|w|-1}$ zugewiesen (alle Vorkommen des Stammes werden auf die Parsemöglichkeiten gleichmäßig aufgeteilt)

3. Die Qualität jedes Parses wird durch die Formel

$$H(\text{stamm/suffix}) = -(|\text{stamm}| \cdot \log(\text{freq}(\text{stamm})) + |\text{suffix}| \cdot \log(\text{freq}(\text{suffix}))$$

bzw. genereller:

$$H(\text{stamm/suffix}) = -|\text{morphem}_i| \cdot \log \text{freq}(\text{morphem}_i)$$

4. Ein Normalisierungswert Z (= die Summe aller Parses P_i) wird erstellt mit $Z = \sum_i e^{H(S_i)}$

5. Jedem Split wird eine Wahrscheinlichkeit $\frac{1}{Z} \cdot e^{H(S_i)}$ zugewiesen.

6. Für jedes Wort wird der beste Parse festgestellt

7. Wiederhole, bis kein Wort seinen optimalen Parse ändert (5 Durchgänge)

Strategie 2: N-gram MI („*m*utual *i*nformation“)

Vorteil: sehr viel schnellere Konvergenz

1. Wir nehmen an, dass jedes Wort mit einem Finalzeichen endet (traditionellerweise „#“)

2. Alle k-gramme für k von 1 bis 6, die am Wortende auftauchen, werden registriert (ausgehend von der Annahme, dass es keine Morpheme gibt, die länger als sechs Buchstaben sind).

3. Den k-grammen $n_1 n_2 ... n_k$ wird eine Wahrscheinlichkeit zugewiesen:

$$\frac{[n_1 n_2 ... n_k]}{Gesamtzahl von k-grammen} \cdot \log \frac{[n_1 n_1 ... n_k]}{[n_1][n_2]...[n_k]}$$

4. Die besten hundert k-gramme werden als Kandidaten behandelt.

5. Alle Wörter werden aufgrund dieser Kandidaten geparst.

6. Hat ein Wort mehr als einen möglichen Parse, wird die Bewertung von Strategie 1 benutzt, um den besten herauszufinden.

Anmerkung: Strategie 1 liefert viele Splits in Präfix+Stamm

Signaturen

Alle Splits werden in Signaturen angeordnet: Eine Struktur paarweise austauschbarer Elemente:

$$\left\{ \begin{array}{l} stamm_1 \\ stamm_2 \\ stamm_3 \end{array} \right\} \left\{ \begin{array}{l} suffix_1 \\ suffix_2 \end{array} \right\}$$

Aufräumen der Signaturen:

Alle Signaturen mit nur einem Stamm (90 %). Alle Signaturen mit nur einem Suffix.

MDL-Modell

Status Optimaler Anfangssplit, teilweise richtige Stamm/Suffix-Zerlegung

Problem Bewertungsschema, um die beste Morphologie als Optimierungsproblem zu erfassen

Lösung **M**inimum **D**escription **L**ength

Eine Methode, die es ermöglicht, all Information einer Morphologie quantitativ zu messen.

Eigenschaften des MDL-Modells

- Basiert auf dem Logarithmus der Frequenz multipliziert mit der Länge in Buchstaben ⇒ Frequenz eines Buchstaben ist eine schlechte Abschätzung mit welcher Wahrscheinlichkeit dieser Buchstabe ein Morphem ist
- Die Morpheme eines Sprachkorpus zu finden, kann als Suche nach der optimalen Kompression gesehen werden, aber nur, wenn dabei nicht die morphologische Komponente mit der reinen Buchstabenkomponente vermischt wird.
- Drei Unterberechnungen: die Wahrscheinlichkeit einer beliebigen Signatur, die Wahrscheinlichkeit eines gegebenen Stammes, die Wahrscheinlichkeit eines gegebenen Suffixes in der Signatur.
- Jede Morphologie hat insgesamt 1.0 an Wahrscheinlichkeit auf ihre Komponenten zu verteilen.

Die „Länge“ einer Morphologie

Notationen

W	Menge aller Wörter
W_{simple}	Alle Wörter mit stamm+suffix oder nur stamm+NULL
W_{complex}	Alle Wörter, deren Stamm selbst zerlegbar ist
f	Suffix
t	Stamm
σ	Signatur
$[W]$	Anzahl der Wörter (token-count)
$< \text{stämme} >$	Anzahl der Stämme (type-count)
A	Anzahl der Buchstaben im Alphabet des Korpus

Die „Länge“ einer Morphologie ist die Summe der Länge der Suffixe, Stämme und Signaturen plus Organisationsaufwand (in einer graphischen Struktur repräsentiert durch Pfeile (=Pointer)).

Die Suffixkomponente

Besteht aus eine Liste von Pointern auf die Suffixe der Sprache und aus der Liste selbst.

Der Pointer selbst hat die Länge $\log \frac{[W]}{[f]}$.

Die Länge jedes Suffix-Eintrags ist direkt proportional zur Anzahl der Buchstaben im Suffix mal einen Proportionalitätsfaktor $\lambda = \log(A)$.

$$\lambda \cdot |f| + \log \frac{[W_A]}{[f]}$$

Die Stammkomponente

Analog:

$$\lambda \cdot |\text{stamm}(w_i)| + \log \frac{[W]}{[\text{stamm}(w_i)]}, w \in W_{\text{un}} \cup W_{\text{simple}}$$

Die Signaturkomponente

1. Liste von Zeigern auf jede Signatur mit der Länge $\log \frac{[W]}{[\sigma]}$
2. Für jede Signatur:
   (a) Anzahl der Stämme in der Signatur
   (b) Anzahl der Suffixe in der Signatur
   (c) Zeiger auf jeden Stamm, bestehend aus:

i. simple/complex-Flag der Länge $\log \frac{[W]}{[W_{\text{simple}}]}$ bzw. $\log \frac{[W]}{[W_{\text{complex}}]}$

ii. Stamm-Zeiger mit $\log \frac{[W]}{[t]}$ für simple-Stämme und

$$\log \frac{[W]}{[\sigma]} + \log \frac{[\sigma]}{[t]} + \log \frac{[\sigma]}{[\text{wörter}(f) \cap \text{wörter}(\sigma)]}$$

(Wörter, die mit dem Suffix enden *und* zur Signatur gehören)

(d) Zeiger auf jedes Suffix mit der Länge $\log \frac{[\sigma]}{[f \in \sigma]}$

Notation

W_{raw}	Menge aller Wörter, die im Korpus tatsächlich vorkommen
W	Menge aller Wörter, die im Korpus vorkommen plus „virtuelle“ Wörter plus morphologische Zusammensetzungen („savings“ und „saving“ werden als zwei Wörter gezählt)

Komprimierte Länge des Korpus

Jedes Wort wird von drei Zeigern repräsentiert:

1. Zeiger auf seine Signatur mit der Länge $\log \frac{[W]}{[\sigma]}$
2. Zeiger auf dem Stamm mit $\log \frac{[\sigma]}{[t]}$
3. Zeiger auf das Suffix mit $\log \frac{[\sigma]}{[f \in \sigma]}$

$$[w]_{\text{raw}} \left[\log \frac{[W]}{[\sigma(w)]} + \log \frac{[\sigma(w)]}{[t(w)]} + \log \frac{[\sigma(w)]}{[f(w) \cap \sigma(w)]} \right], w \in W$$

Damit haben wir eine quantitative Beschreibung durch MDL, die wir durch bestimmte Algorithmen zu verkleinern versuchen können.

Von den Kandidaten einer Analyse werden die ausgewählt, die eine Verkleinerung der Beschreibungslänge herbeiführen.

Heuristiken für Morphologiekandidaten

Suffixe aufteilen

Suffixe wie „ments“ und „ings“ sollen in „ing/ment“ + „s“ zerlegt werden

1. alle Signaturen, die aus einem einzelnen einbuchstabigen Suffix bestehen, werden entfernt (was auch gültige Suffixe betrifft, aber siehe peripheral cut)
2. Suffixe einer Signatur werden untersucht, ob sie mit dem gleichen Buchstaben/ der gleichen Buchstabensequenz beginnen (te.ting.ts)
3. Es wird evaluiert, ob die Beschreibungslänge sich verkleinert, wenn statt dessen e.ing.s verwendet

Triage (= Sortierung nach Schwere)

Signaturen, die wenige Stämme oder ein einzelnes Suffix enthalten, werden inspiziert

⇒ Wann ist ein Datensatz reich genug, um die Existenz einer linguistischen Signatur zu rechtfertigen?

Soll es eine Signatur .hood geben, auch wenn die Stämme dieser Signatur mit keiner anderen Endung vorkommen?

Rechtfertigt das vorkommen von look, book, loot und boot in einem Korpus eine Signatur

$$\left\{ \begin{array}{c} loo \\ boo \end{array} \right\} \left\{ \begin{array}{c} t \\ k \end{array} \right\}$$

?

Zwei Möglichkeiten mit MDL:

- Erreicht man eine Kompression, wenn man die Signatur auflöst und die enthaltenen Wörter als einfach behandelt (implementiert)?
- Untersignaturen werden gebildet und mit den vorhandenen Signaturen abgeglichen (NULL.ine.ly nur mit „just“, zu NULL.ly + NULL.ine (def, eng etc.)) → würde in machen Fällen nach sich ziehen, dass ein einzelner Stamm zu mehreren Signaturen gehören kann

Führt zum Teil zum Verlust sehr guter Signaturen, z. B. NULL.ness, NULL.ful, NULL.ish

Andere Methoden sind noch experimentell, z. B. Wortpaar-Substitutions-Tests. Könnten zu Mustern wie a→ä o→ue führen

Fehler

- Eigennamen werden analysiert „Anab-el“, „Alexand-er“
- Reduplikation und orthographische Veränderungen werden nicht erkannt: abet/abetted/abetting führt zu NULL.ted.ting statt zu NULL.ed.ing + Vorhersagbarkeit von Doppel-t

Verbesserungen

1. Allomorphe identifizieren
2. Paradigmen aus Signaturen ableiten. NULL.ed.ing z. B. soll automatisch als Teil von NULL.ed.ing.s erkannt werden. Besonderheit: NULL.s sollte eher von 's.NULL.s als von NULL.ed.s abgeleitet werden
3. Präfixbehandlung
4. Zusammengesetze Wörter identifizieren
5. Suffix-Suffix-Beziehungen
6. Substraktive Morpheme: NULL.ed.ing.s und e.ed.es.ing sollen zusammgeführt werden (love bzw. jump)