Praktikum Verarbeitung natürlicher Sprache

Parsing von probabilistische kontextfreien Grammatiken

Richard Mörbitz Heiko Vogler

16. April 2021

1. Eckdaten

1.1. Einbringung ins Studium

Umfang: 0/0/4 SWS (6 LP, 180 h Arbeitsaufwand)

Dauer: 15 Wochen

Module und Prüfungsmodi:

Modul	Name der Lehrveranstaltung	Prüfung
INF-B-510	Praktikum Verarbeitung natürlicher Sprache (6 LP, 180 h, deckt 50 % des Moduls ab auffüllen *nicht* mit einem Praktikum möglich)	mündlich (für Praktikum)
INF-B-520	Praktikum Verarbeitung natürlicher Sprache (6 LP, 180 h, deckt 50 % des Moduls ab auffüllen *nicht* mit einem Praktikum möglich)	mündlich (für Praktikum)
INF-MA-PR	Master-Praktikum Verarbeitung natürlicher Sprache (6 LP, 150 h, deckt 50 % des Moduls ab)	Kolloquium (für Praktikum)
INF-PM-FPG	Forschungsprojekt Verarbeitung natürlicher Sprache (6 LP, 180 h, deckt 50 % des Moduls ab)	Kolloquium (gesamtes Modul)

Abgrenzung:

- Das Praktikum passt nicht ins Modul INF-D-940, da das Modul zweigeteilt ist (2 SWS Praktikum/etc. + 2 SWS Seminar). Es besteht die Option, das Praktikum mit verringertem Umfang (2 SWS) zu absolvieren.
- Das Praktikum passt nicht ins Modul INF-PM-FOR, da das Modul dreigeteilt ist (2 SWS Vorlesung + 2 SWS Übung/Seminar + Selbststudium).

1.2. Voraussetzungen

- Grundlegendes theoretisches Verständnis und sicherer Umgang mit kontextfreien Grammatiken, wie sie in der Vorlesung Formale Systeme (Modul INF-B-270) vermittelt werden.
- Sicherer Umgang mit einer geläufigen Programmiersprache (C, C++, Python, Haskell, Rust, Java).

2. Aufgabenbeschreibung

Probabilistische kontextfreie Grammatiken. In der Verarbeitung natürlicher Sprache wird die Zugehörigkeit eines Wortes zu einer Sprache oftmals nicht als Wahrheitswert, sondern als reelle Zahl zwischen 0 und 1 formuliert. Man erhält damit einen *Grad der Zugehörigkeit* des Wortes zur Sprache. So können z.B. häufige von seltenen sprachlichen Konstruktionen unterschieden werden. In einer probabilistische kontextfreien Grammatik (kurz: PCFG) wird zu diesem Zweck jeder Regel eine reelle Zahl zwischen 0 und 1 zugeordnet (die *Wahrscheinlichkeit* der Regel), sodass die Summe der Wahrscheinlichkeiten aller Regeln mit der gleichen linken Seite 1 ergibt. Die Wahrscheinlichkeit einer Ableitung erhält man durch Multiplikation der Regelwahrscheinlichkeiten und die Wahrscheinlichkeit eines Wortes durch Addition der Wahrscheinlichkeiten aller Ableitungen dieses Wortes.

Parsing. Unter Parsing versteht man die Berechnung einer syntaktischen Struktur eines gegebenen Satzes w in einer natürlichen Sprache $L(G)$. Diese syntaktische Struktur wird üblicherweise als Baum angegeben, dem sogenannten *Parsebaum*. Wenn die natürliche Sprache $L(G)$ durch eine (probabilistische) kontextfreie Grammatik G gegeben ist, dann entspricht jeder Parsebaum einer Linksableitung von w in G.

Probabilistisches Parsing. Grammatiken für natürliche Sprachen sind in der Regel *mehrdeutig*, d.h. es existieren mehrere Parsebäume für denselben Satz. Die Wahrscheinlichkeit eines Parsebaums ist das Produkt der Wahrscheinlichkeiten aller zu seiner Ableitung benutzten Regeln. Beim probabilistischen Parsing wird Mehrdeutigkeit mithilfe der Wahrscheinlichkeiten der Parsebäume aufgelöst. Typische Problemstellungen sind *Best-Parsing*, das Finden eines Parsebaums mit der höchsten Wahrscheinlichkeit, und *n-best-Parsing*, das Finden von n Parsebäumen mit der höchsten Wahrscheinlichkeit.

Baumkorpora und Grammatikinduktion. Ein Baumkorpus ist eine Multimenge von Parsebäumen einer natürlichen Sprache. Baumkorpora werden von Linguisten mithilfe spezieller Annotationswerkzeuge manuell oder halbautomatisch aus verfügbaren Texten (z.B. Zeitschriftentexte oder Parlamentsmitschriften) erstellt. Sie können z.B. zur Gewinnung einer probabilistischen kontextfreien Grammatik genutzt werden.

2.1. Aufgaben

Die Bearbeitung von 3 als *optional* gekennzeichneten Aufgaben ist obligatorisch.

(1) Implementierung eines Algorithmus zur Grammatikinduktion

- Eingabe: Baumkorpus im *Penn Treebank* (PTB) Format:

  `(ROOT (S (ADV-MO Auch) (PTKNEG-NG nicht) (NP-SB (ART-NK die) (NN-NK Polizei))) ($. .))`

  Jeder Baum steht auf einer eigenen Zeile.
- Ausgabe: induzierte PCFG im Format des Berkeley-Parsers. Grammatiken, die aus Korpora im PTB-Format induziert werden, sind terminal-separiert, d.h. auf der rechten Seite jeder Regel steht entweder eine Sequenz von $n > 0$ Nichtterminalen oder genau ein Terminal. Grammatiken werden wie folgt abgespeichert, wobei `grammar` der Name der Grammatik ist:
  - jede Regel ρ der Form $A \to A_1 \dots A_n$ mit $P(\rho) = w$ wird in eine eigene Zeile in die Datei `grammar.rules` gespeichert:

    `A -> A_1 ... A_n w`

    Nichtterminale dürfen keine Leerzeichen enthalten. Leerzeichen trennen jeweils Nichtterminale, den Pfeil und die Wahrscheinlichkeit w.
  - jede lexikale Regel ρ der Form $A \to$ `v` mit $P(\rho) = w$ wird in eine eigene Zeile in die Datei `grammar.lexicon` gespeichert:

    `A v w`

  - jedes Terminal `v` wird in eine eigene Zeile in `grammar.words` gespeichert

    `v`
- Die Wahrscheinlichkeiten der Grammatikregeln werden durch relative Häufigkeitsschätzung (RFE) bestimmt.

(2) Implementierung eines Algorithmus zum Parsing

- Eingabe: binarisierte PCFG G, d.h. für jede Regel $A \to A_1 \dots A_n$ gilt $n \leq 2$, und eine Sequenz von wie folgt vorverarbeiteten Sätzen:
  - jeder Satz steht auf einer Zeile
  - Terminale (d.h. Wörter und Punktuationszeichen) sind durch Leerzeichen voneinander getrennt
  - Sonderzeichen (Klammern, Leerzeichen innerhalb von Terminalen) sind durch Escape-Sequenzen ersetzt
- Ausgabe: Sequenz von Bäumen im PTB-Format; zu jedem Eingabesatz w der beste Parsebaum von w in G oder `(NOPARSE` w`)` falls keiner existiert

Als zugrundeliegender Algorithmus wird entweder Bottom-Up-Parsing (CYK) *oder* Best-First-Parsing (Knuths Algorithmus) implementiert.

(3) Implementierung von Korpustransformationen

(3 a) Debinarisierung (alle durch Binarisierung eingefügten Knoten mit der Form `A|<A1,A2,A3,...>` werden entfernt)

(3 b) triviales *Unking* (jedes Wort im Korpus, das weniger als n mal vorkommt, wird durch das Symbol `UNK` ersetzt). Das Unking ist auch im Parser zu reflektieren.

(3 c) *(optional)* Binarisierung und Markovisierung (Einfügen zusätzlicher Knoten, um die Binarisierungseigenschaft herzustellen) [5]

(3 d) *(optional)* Smoothing der induzierten Grammatik (Verbesserung der Generalisierung durch bspw. Ersetzung seltener Wörter durch Symbole, z.B. [2, 1])

(4) Implementierung von Optimierungen des Parsers

(4 a) *(optional)* Pruning (Beschränkung der untersuchten Spans) durch

- Beam-Search auf gesamter Agenda (nur Best-First-Parsing) *oder*
- Beam-Search pro Zelle des Charts (nur Bottom-Up-Parsing),

Implementierung durch Beam konstanter Größe (maximale Zahl gleichzeitig berücksichtigter Spans) *oder* Threshold-Beam (Verwerfen von Spans, deren Wahrscheinlichkeit einen Schwellwert unterschreitet), auch Kombination möglich

(4 b) *(optional)* k-Best (Ausgabe von k besten Parsebäumen für einen zur Laufzeit übergebenen Parameter k statt nur des besten Parsebaums) [3]

(4 c) *(optional)* A^*-Suche (Nutzung des Viterbi Inside/Outside-Weights als Heuristik zur Auswahl des nächsten Spans aus der Agenda) [4]. Die Viterbi Outside-Weights werden in einer Datei `grammar.outside` in folgendem Format gespeichert: Je Nichtterminal `A` mit Outside-Weight `w` enthält die Datei eine Zeile der Form

```
A w
```

Im Falle von Bottom-Up-Parsing kann diese Aufgabe auch mit Pruning kombiniert werden, wobei das für die Pruningentscheidung maßgebliche Gewicht eines Items als Produkt aus Viterbi Outside-Weight des Nichtterminals und Wahrscheinlichkeit der besten Ableitung für das Item berechnet wird.

2.2. Kriterien für die Vergabe der Leistungspunkte

- Aufgabe (1): Grammatikinduktion muss weiterverwendbare Ausgabe liefern
- Aufgabe (2): Parser muss einen Parse von w in G ausgeben (sofern einer existiert)
- Aufgabe (3): mindestens eine der beiden Pflichtaufgaben ist erfüllt

- Kolloquium / mündliche Prüfung: muss absolviert werden

Für das Erreichen von guten bzw. sehr guten Noten müssen alle Pflichtanforderungen innerhalb des vorgegebenen Zeitrahmens voll erfüllt und die wahlobligatorischen Aufgaben hinreichend bzw. vollständig bearbeitet worden sein.

3. Ablauf

Im Folgenden steht VW für Vorlesungswoche. Eine Vorlesungswoche ist eine Woche des Semesters, in der Vorlesungen stattfinden können.

19. 04. Einführungsveranstaltung/Tutorium 1 (Organisatorisches, Überblick über PCFG-Parsing, Induktionsmechanismus), Aufgabenstellung Aufgabe (1)

2.–4. VW selbstständige Bearbeitung der Aufgabe (1)

05. 05. Abgabe

10. 05. Tutorium 2 (Auswertung vorherige Aufgabe, Vorstellung beider Parsing-Algorithmen, Datenstrukturen), Aufgabenstellung Aufgabe (2)

5.–9. VW selbstständige Bearbeitung der Aufgabe (2)

09. 06. Abgabe

14. 06. Tutorium 3 (Auswertung vorherige Aufgabe, Vorstellung der Korpustransformationen), Aufgabenstellung Aufgabe (3)

21. 06. Tutorium 4 (Vorstellung der Parseroptimierungen), Aufgabenstellung Aufgabe (4)

10.–14. VW selbstständige Bearbeitung der Aufgaben (3) und (4)

14. 07. Abgabe

19. 07. Tutorium 5 (Auswertung optionale Aufgaben, Wettbewerb)

4. Wettbewerb

Der Wettbewerb wird zum Ende der 14. VW – nach der Abgabe – unter allen Einreichungen durchgeführt (die Teilnahme ist fakultativ). Die Ergebnisse werden im anschließenden, letzten Tutorium vorgestellt und diskutiert.

- 2 Kategorien: Accurracy und Speed
- Jury kann Parser disqualifizieren (wenn z.B. nur Dummy-Bäume ausgegeben werden, um die Speed-Kategorie zu gewinnen)
- Ranking durch automatisierte Tests

5. Technische Umsetzung

Der Code muss im PC-Pool APB/E067 (Ubuntu 18.04) laufen. Alle Schritte zur Erstellung des Programms sollen durch den Aufruf `make` erfolgen. Wir geben ein Kommandozeileninterface vor (siehe Anhang A). Die Bewertung erfolgt durch automatisierte Tests (Integration-Test). Zur Überprüfung, ob Programmaufruf und Ausgabeformats korrekt implementiert sind, stellen wir Tests zur Verfügung.

A. Kommandozeilenschnittstelle

Die Abgabe beinhaltet eine ausführbare Datei mit dem Namen `pcfg_tool`. Die Implementierung der Teilaufgaben wird über Subkommandos aufgerufen (ähnlich wie bei `git`). Teilaufgaben können als eigenständige Programme umgesetzt werden, in diesem Fall muss `pcfg_tool` die Kommandozeilenargumente und Standardeingabe der betreffenden Subkommandos an die jeweiligen Programme weiterleiten.

Einige Argumente betreffen nur wahlobligatorische Aufgaben, diese wird dann in Klammern angegeben. Wurde eine wahlobligatorische Aufgabe nicht gelöst, dann soll das Programm bei Vorhandensein diese Aufgabe betreffender Argumente mit dem Status 22 terminieren. Der Parser soll standardmäßig ohne Optimierungen laufen und keine Korpustransformationen außer Binarisierung annehmen. Optimierungen und Korpustransformationen werden explizit über Flags angesteuert.

Sequenzen von Bäumen und Sätzen werden stets zeilenweise ein- und ausgegeben, d.h. es befindet sich genau ein Objekt pro Zeile. Die Programmausgabe muss die Reihenfolge der Eingabeobjekte beibehalten.

```
pcfg_tool [COMMAND]
  Tools zum PCFG-basierten Parsing natürlichsprachiger Sätze

pcfg_tool induce [GRAMMAR]
  Liest eine Sequenz Konstituentenbäume von der Standardeingabe
  und gibt eine aus diesen Bäumen induzierte PCFG auf der
  Standardausgabe aus. Wird das optionale Argument GRAMMAR
  angegeben, dann wird die PCFG stattdessen in den Dateien
  GRAMMAR.rules, GRAMMAR.lexicon und GRAMMAR.words gespeichert
  (s. (1)).

pcfg_tool parse [OPTIONS] RULES LEXICON
  Liest eine Sequenz  natürlichsprachiger Sätze von der Standard-
  eingabe und gibt die zugehörigen besten Parsebäume im PTB-Format
  bzw. (NOPARSE <Satz>) auf der Standardausgabe aus. RULES und
  LEXICON sind die Dateinamen der PCFG (s. (1)).

  -p --paradigma=PARADIGMA
    Parserparadigma (cyk oder deductive).
```

```
    Default: Wahl in (2)
  -i --initial-nonterminal=N
    Definiere N als Startnichtterminal.
    Default: ROOT
  -u --unking
    Ersetze unbekannte Wörter durch UNK
  -s --smoothing
    Ersetze unbekannte Wörter gemäß der Smoothing-Implemen-
    tierung (3 d)
  -t --threshold-beam=THRESHOLD
    Führe Beam-Search durch mit Threshold (4 a).
  -r --rank-beam=RANK
    Führe Beam-Search durch mit Beam konstanter Größe (4 a).
  -k --kbest=K
    Gib K beste Parsebäume statt des besten aus (4 b).
  -a --astar=PATH
    Führe A*-Suche durch (4 c). Lade die Outside weights aus der
    Datei PATH.

pcfg_tool binarise [OPTIONS]
  Liest eine Sequenz Konstituentenbäume von der Standardeingabe
  und gibt die entsprechenden binarisierten Konstituentenbäume
  auf der Standardausgabe aus (3 c).

  -h --horizontal=H
    Horizontale Markovisierung mit H.
    Default: unendlich (999)
  -v --vertical=V
    Vertikale Markovisierung mit V.
    Default: 1

pcfg_tool debinarise
  Liest eine Sequenz (binarisierter) Konstituentenbäume von der
  Standardeingabe und gibt die ursprünglichen (nicht
  binarisierten) Konstituentenbäume auf der Standardausgabe aus.

pcfg_tool unk [OPTIONS]
  Liest eine Sequenz Konstituentenbäume von der Standardeingabe
  und gibt die durch triviales Unking erhaltenen Bäume auf der
  Standardausgabe aus.

  -t --threshold=T
    Schwellwert der absoluten Häufigkeit für das Unking.
```

```
pcfg_tool smooth [OPTIONS]
  Liest eine Sequenz Konstituentenbäume von der Standardeingabe
  und gibt die durch Smoothing erhaltenen Bäume auf der
  Standardausgabe aus (3 d).

  -t --threshold=T
    Schwellwert der absoluten Häufigkeit für das Unking.

pcfg_tool outside [OPTIONS] RULES LEXICON [GRAMMAR]
  Berechnet Viterbi Outside weights für jedes Nichtterminal
  der Grammatik und gibt diese auf der Standardausgabe aus.
  Wird das optionale Argument GRAMMAR angegeben, dann werden
  die Outside weights in die Datei GRAMMAR.outside
  gespeichert (4 c).

  -i --initial-nonterminal=N
  Definiere N als Startnichtterminal.
  Default: ROOT
```

Literatur

[1] Peter F. Brown u. a. "Class-based N-gram Models of Natural Language". In: *Comput. Linguist.* 18.4 (Dez. 1992), S. 467–479. ISSN: 0891-2017. URL: http://dl.acm.org/citation.cfm?id=176313.176316.

[2] Daniel Dakota. "The Devil is in the Details: Parsing Unknown German Words". In: *Language Technologies for the Challenges of the Digital Age*. Hrsg. von Georg Rehm und Thierry Declerck. Cham: Springer International Publishing, 2018, S. 23–39. ISBN: 978-3-319-73706-5. DOI: 10.5555/176313.176316.

[3] Liang Huang und David Chiang. "Better K-best Parsing". In: *Proceedings of the Ninth International Workshop on Parsing Technology*. Parsing '05. Vancouver, British Columbia, Canada: Association for Computational Linguistics, 2005, S. 53–64. URL: http://dl.acm.org/citation.cfm?id=1654494.1654500.

[4] Dan Klein und Christopher D. Manning. "A* Parsing: Fast Exact Viterbi Parse Selection". In: *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*. 2003. URL: http://aclweb.org/anthology/N03-1016.

[5] Dan Klein und Christopher D. Manning. "Accurate Unlexicalized Parsing". In: *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*. ACL '03. Sapporo, Japan: Association for Computational Linguistics, 2003, S. 423–430. DOI: 10.3115/1075096.1075150.