Mit Maximum-Entropie das Parsing natürlicher Sprache erlernen

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

4. Juli 2016
Marc Schreiber, M. Sc.

Inhaltsverzeichnis

1 Einleitung

Viele Anwendung verwenden das World Wide Web als Datenquelle, um dem Benutzer aufbereitete Informationen zu präsentieren. Beispielsweise hat Google den Knowledge-Graph [37] in die Google-Suche integriert, um bei der Suche nach der Person *Satya Nadella* nicht nur eine Liste von Webseiten, die *Satya Nadella* im Text aufführen, zu zeigen, sondern auch um Fakten zur Person darzustellen (vgl. Abbildung 1). Diese Fakten stammen aus dem Knowledge-Graph, der einen Wissensgraphen repräsentiert. Dieser wird aus Daten generiert, die in natürlich-sprachlichen Texten des World Wide Webs verfasst sind[1]. Die Extraktion von Informationen aus natürlichen Sprachen gehört zur Disziplin der natürlichen Sprachverarbeitung (auch Computerlinguistik genannt).

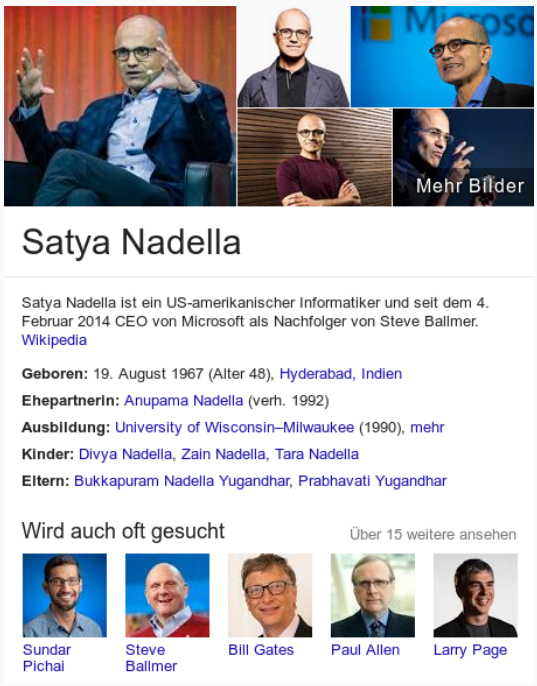


Abbildung 1: Darstellung des Suchergebnisses mit Hilfe des Google-Knowledge-Graphs

Um solch einen Wissensgraphen aus natürlich-sprachlichen Texten zu generieren, müssen Texte ausgehend von ihrer sequentiellen Reihenfolge der enthaltenen UTF8-Zeichen in einen Graphen transformiert werden. Während dieser Transformation wird eine Reihe an Werkzeugen eingesetzt, die stufenweise den Text mit Strukturinformationen anreichern. In der ersten Stufe wird ein Text in Einheiten (Sätze und Wörter/Satzzeichen) zerlegt und darauf aufbauend werden die Wortarten für die Wörter/Satzzeichen bestimmt. In der darauf-aufbauenden Stufe werden Sätze mit Parse-Trees/Ableitungsbäumen angereichert.

Die letztgenannte Stufe wird als Parsing bezeichnet. Das Parsing der natürlichen Sprachverarbeitung bestimmt für jeden gegebenen Satz, der in Wörter und Satzzeichen unterteilt

[1] Der Google-Knowledge-Graph basiert neben Daten aus natürlich-sprachlichen Texten ebenfalls auf anderen Datenquellen, wie z. B. DBpedia [34].

und mit Wortarten versehen ist, einen Ableitungsbaum. Dieser Ableitungsbaum ist vergleichbar mit einem Abstract-Syntax-Tree [1] aus dem Compilerbau. Der Abstract-Syntax-Tree stellt den Quelltext eines Computerprogramms als Baumstruktur dar. Ableitungsbäume werden sowohl im Compilerbau als auch in der natürlichen Sprachverarbeitung genutzt, um weitere Operationen darauf anzuwenden.

Im Compilerbau wird basierend auf dem Abstract-Syntax-Tree eine semantische Überprüfung des Quelltextes durchgeführt [1, Kapitel 1.2, Seite 8]. In der natürlichen Sprachverarbeitung können mit Hilfe von Ableitungsbäumen in Kombination mit Named-Entity-Recognition (NER) sprachliche Relationen in Texten erkannt werden. Dieses Verfahren ist in Abbildung 2 auf der nächsten Seite skizziert: Wenn der Satz *„Satya Nadella works for Microsoft"* aus dem World Wide Web verarbeitet wird (Abbildung 2a), dann wird zunächst in mehreren Verarbeitungsstufen der Ableitungsbaum sowie weitere Informationen generiert (Abbildung 2b). Basierend darauf werden Relationen extrahiert (Abbildung 2c) und einem Wissensgraphen hinzugefügt.

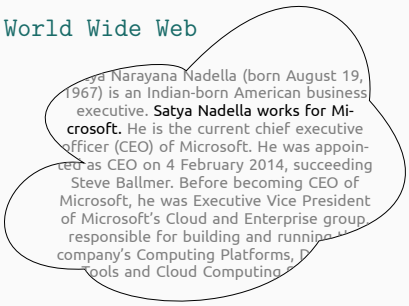
Im Compilerbau werden formale Sprachen [13] eingesetzt, um basierend auf den entsprechenden formalen Grammatiken Parser zu erstellen, die das Wortproblem [27, Kapitel 1.1.3] lösen und dabei den Abstract-Syntax-Tree generieren. Bei der Verarbeitung natürlicher Sprache können formale Grammatiken in der Regel nicht eingesetzt werden:

- Es gibt keine allgemeine Formalisierung von natürlicher Sprache und daher existiert keine formale Grammatik, die zum Parsen eingesetzt werden kann. Mehrere Studien untersuchen natürliche Sprachen bzgl. der Einordbarkeit in die Chomsky-Hierarchie [31, Kapitel 6.1]: Englisch kann nicht als kontextfreie Sprache formalisiert werden [3, 12, 23]; Schweizer-Deutsch gilt als mindestens kontextsensitive Sprache [30].

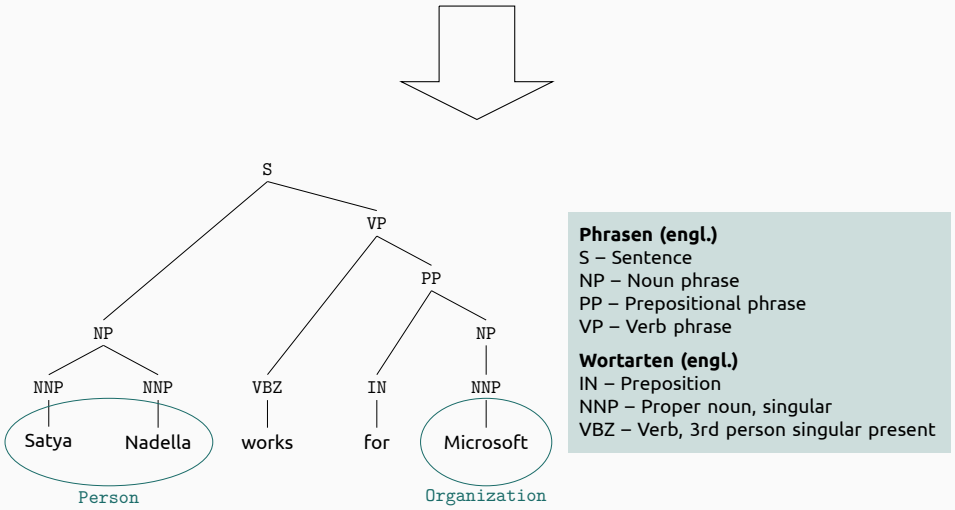
- Es gibt Sprachen, wie z. B. Attempto Controlled English [8], die als Teilmenge einer natürlichen Sprache eine formale Sprache bilden. Diese formale Sprache erscheint als natürliche Sprache für den entsprechenden Muttersprachler und kann vom diesen verstanden werden. Diese Sprachen finden jedoch nur in wenigen Domänen Anwendung und besitzen eine geringe Verbreitung im World Wide Web. Für solche natürliche Sprachen können Parsing-Verfahren des Compilerbaus verwendet werden[2].

- Gebe es eine formale Grammatik für natürliche Sprachen, dann würden grammatikalisch falsche Sätze nicht als Teil der Sprache erkannt und somit herausgefiltert. Diese Sätze können jedoch von einem Menschen verstanden werden. In Bezug auf den Inhalt können wichtige Informationen in diesen Sätzen enthalten sein und die Informationen gingen in der skizzierten Anwendung (vgl. Abbildung 2) verloren.

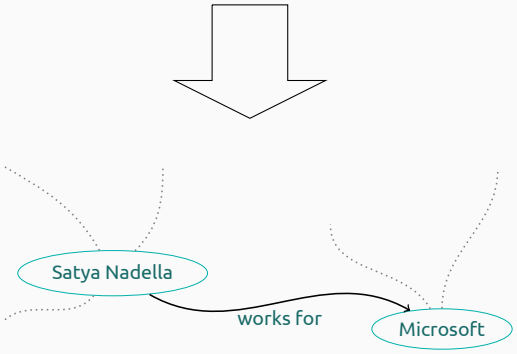
[2] Eysholdt und Behrens [7] bieten mit Xtext ein Werkzeug, um solche domänenspezifischen Sprachen zu parsen.



(a) Das World Wide Web als Informationsquelle



(b) Ableitungsbaum mit Named-Entities (NEs)



(c) Graph-Darstellung der erkannten Relation

Abbildung 2: Informationsgewinnung durch natürliche Sprachverarbeitung

2 Der Weg von formalen Methoden über die Korpuslinguistik zur heutigen natürlichen Sprachverarbeitung

Die ersten Anwendungen, die natürliche Sprache verarbeiten, verwenden feste Regelmengen und Wörterbücher. Weizenbaum [32] und Winograd [41] verwenden in ihren Programmen ELIZA und SHRDLU solche Regeln, die in Lisp formuliert sind, um die Möglichkeiten der Kommunikation zwischen Computern und Menschen über natürliche Sprache zu verdeutlichen[3]. Das folgende Beispiel illustriert, wie ELIZA mit Regelwerk und Wörterbuch arbeitet (vgl. [35]).

In dem Beispieleingabesatz *„Ich habe ein Problem mit meinem Vater"* erkennt ELIZA *Vater* als Unterbegriff von *Familie*. Aufgrund einer Regel antwortet ELIZA: *„Erzählen Sie mir mehr über Ihre Familie"*. Da das Regelwerk nicht vollständig ist, antwortet ELIZA auf den Satz *„Krieg ist der Vater aller Dinge"* mit dem selben vorherigen Antwortsatz. Findet das Programm keine Antwort über eine Regel, antwortet es mit Ausweichphrasen: *„Davon verstehe ich nichts, lassen Sie uns über ein anderes Thema reden."*

Bei solchen Systemen stellt sich jedoch heraus, dass diese Regelwerke schnell an ihre Grenzen stoßen, und es ist nicht garantiert, dass solche Regelwerke minimal und/oder vollständig sind. Mit den folgenden Arten der Mehrdeutigkeiten wird veranschaulicht, warum dieser regelbasierte Ansatz scheitert.

Syntaktische Mehrdeutigkeiten bedeutet, dass ein einzeln betrachteter Satz mehrere Ableitungsbäume besitzen kann, die verschiedene Bedeutungen induzieren. In Abbildung 3 auf der nächsten Seite sind die möglichen Ableitungsbäume für den Satz *„Der Mann beobachtet das Mädchen mit dem Fernglas"* gegeben. Die beiden Ableitungsbäume geben jeweils Aufschluss darüber, welche von beiden Personen das Fernglas trägt (vgl. Abbildung 3a und 3b).

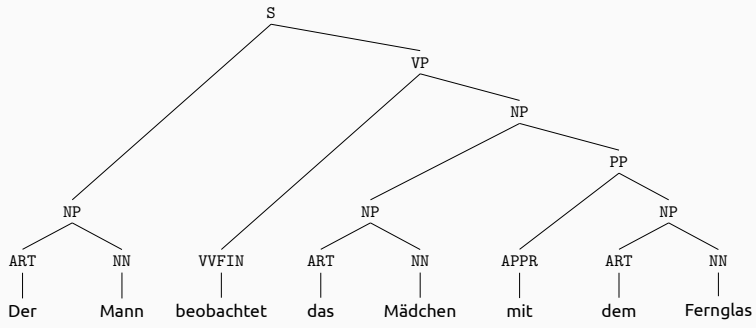
Semantische Mehrdeutigkeiten bedeutet, dass ein einzeln betrachteter Satz mehrere Deutungsweisen bei gleichem Ableitungsbaum aufweist. Abbildung 4 auf Seite 6 zeigt den eindeutigen Ableitungsbaum für den Satz *„Er öffnet das Schloss"*. Das Wort *Schloss* kann hier ein Vorhängeschloss oder ein Bauwerk meinen.

In beiden Fällen benötigt ein Werkzeug (und auch der menschliche Leser) den Kontext, d. h. es werden weitere Informationen benötigt, wie solche Sätze zu interpretieren sind. Im Fall der syntaktischen Mehrdeutigkeit kann der voranstehende Satz wichtige Kontextinformationen liefern: *„Nachdem der Mann mit seinem neuen Fernglas nach Hause kam, setzte er sich ans Fenster."* Über diesen Satz wird eingeleitet, dass der Mann das Fernglas besitzt. Dies zeigt jedoch, dass Informationen des Kontextes nicht rein auf der strukturellen Ebene ermittelt werden können. Hier muss die Bedeutung als Information in das Parsing des folgenden Satz mit einfließen. Da beim Parsen von Sätzen Informationen über den Kontext aus mehreren Ebenen einfließen müssen, ist es u. a. bisher nicht gelungen, deduktiv eine formale Grammatik für natürliche Sprache zu finden.

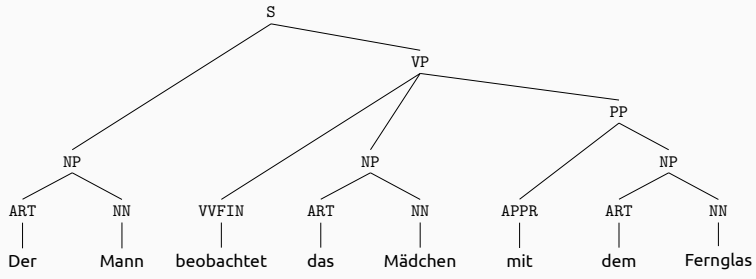
Parallel zur Suche einer formale Grammatik ist die induktive Methode der Korpuslinguistik [19, 22] entstanden. Mittels Textkorpora, die aus einer Sammlung realer Texte bestehen[4], werden Analysen durchgeführt, um bestehende Hypothesen zu be-/widerlegen oder um neue Hypothesen zu einer natürlichen Sprache zu gewinnen. Für Analysen werden die Textkorpora oft manuell annotiert, z. B. wird für jeden Satz im Korpus der entsprechende Ableitungsbaum

[3] Die Regeln von ELIZA können auf Github eingesehen werden: `https://github.com/jeffshrager/elizagen/`

[4] Textkorpora bestehen oft aus Texten einer einzelnen Domäne, z. B. Zeitungsartikel oder Aufsätze der Biomedizin.



(a) Das **Mädchen** hält das Fernglas in der Hand



(b) Der **Mann** hält das Fernglas in der Hand

Abbildung 3: Syntaktisch mehrdeutiger Satz mit mehreren Bedeutungen

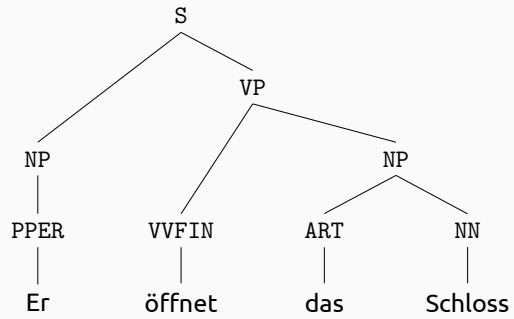
notiert – die Penn-Treebank [21] entspricht einem solchen annotierten Textkorpus. Diese Annotationen sind notwendig, um Aussagen über natürliche Sprache zu be-/widerlegen – Manning [20] nutzt z. B. Annotationen, um zu zeigen, dass bisherige Erkenntnisse der Wortarten zur Wortartbestimmung unzureichend sind.

Der Aufwand für die Erstellung eines annotierten Textkorpus ist sehr groß, weil viele Texte enthalten sind (bis zu ≈1.000.000 Wörter) und das Annotatieren eines einzelnen Satzes aufwendig ist. Um die Arbeit zu erleichtern, sind unterstützende Werkzeuge entstanden, die Annotation vorschlagen: z. B. das Werkzeug von Greene und Rubin [11] schlug ca. 70% aller Wortarten korrekt vor und über eine Dekade konnte ein Korpus von 1.000.000 Wörter nahezu vollständig korrekt annotiert werden[5]. Das Werkzeug von Greene und Rubin [11] nutzt ebenfalls ein festes Regelwerk: z. B. kann auf einen Artikel ein Nomen folgen, jedoch kein Verb.

Die Textkorpora der Korpuslinguistik bieten in Kombination mit Supervised-Machine-Learning [2] einen Vorteil gegenüber der Formalisierung der natürlichen Sprache durch formale Grammatiken: da die Textkorpora mit Annotationen versehen sind, bieten Supervised-Machine-Learning-Algorithmen die Möglichkeit, die Regelwerke bisheriger Tools (vgl. Greene und Rubin [11]) durch automatisch erkannte Regeln zu ersetzen. Dieser Ansatz bietet für die Verarbeitung natürlicher Sprache folgende Vor- und Nachteile:

+ Die Regelwerke müssen für Werkzeuge nicht manuell erstellt werden, was die Entwicklungszeit beschleunigt.

[5] Nahezu vollständig bedeutet, dass in einigen Fällen die menschlichen Annotatoren sich nicht einigen konnte, welches die korrekte Wortart ist.



Phrasen
S – Satz
NP – Nominalphrase
VP – Verbalphrase

Wortarten
ART – Artikel
NN – Nomen
PPER – Personalpronomen
VVFIN – finites Verb

Abbildung 4: Syntaktisch eindeutiger Satz mit mehreren Bedeutungen

+ Die automatisch erstellten Regelwerke sind umfangreicher und arbeiten mit einer höheren Genauigkeit.
+ Die Beschränkung der Textkorpora auf eine Domäne führt in vielen Fällen dazu, dass der Kontext implizit definiert ist, z. B. bei Architekturtexten ist mit hoher Wahrscheinlich bei dem Wort *Schloss* das *Bauwerk* gemeint.
+ Die automatisch erstellten Regelwerke sind robuster gegenüber unbekannten Eingaben (z. B. Sätze die nicht im Korpus existieren) oder auch gegenüber fehlerhaften Sätzen, die in realen Daten häufig existieren.
■ Die automatisch erstellten Regelwerke können zwar auf Texten anderer Domänen angewendet werden (Vorteil), jedoch muss dann mit einer geringeren Genauigkeit gerechnet werden (Nachteil).
– Der Kontext, wird nur teilweise durch die Beschränkung auf eine Domäne aufgelöst, da der Kontext nicht nur durch textuelle Muster bestimmt wird.
– Eine große Menge an Beispielen, die manuell erstellt werden müssen, wird für die Lernphase benötigt.
– Die Qualität des automatischen Regelwerks hängt von der Qualität der Beispiele ab, d. h. qualitativ schlechte Beispiele führen zu qualitativ schlechtem Regelwerk.

3 Supervised-Machine-Learning mit Maximum-Entropie

Supervised-Machine-Learning [2] ist eine Methodik aus dem Bereich des maschinellen Lernens. Dabei erlernen Supervised-Machine-Learning-Verfahren mittels gegebener Beispiele existierende Muster/Gesetzmäßigkeiten innerhalb der Beispiele zu erkennen. Ein Beispiel besteht aus einem Tupel (x, y). Dabei stellt x ein Eingabebeispiel dar und y das dazugehörige Ausgabebeispiel. Durch eine Trainingsphase lernt ein Supervised-Machine-Learning-Verfahren die Gesetzmäßigkeiten zwischen x und y und es ist nach der Trainingsphase in der Lage für unbekannte x ein zugehöriges y zu bestimmen. Die lineare Regression [39] ist ein Beispiel für ein Supervised-Machine-Learning-Verfahren, z. B. $x \hat{=}$ Grundstücksgröße und $y \hat{=}$ Grundstückspreis. In dieser Arbeit wird sich im Folgenden auf die Klassifikation beschränkt, d. h. y entspricht einer bestimmten Kategorie (z. B. entspricht y einem niedrigen oder hohen Grundstückspreis).

Für die Wortartbestimmung[6] der natürlichen Sprachverarbeitung bzgl. des Beispielsatzes *„Satya Nadella works for Microsoft"* entspricht ein Tupel (x_b, y_b) für das Wort *works* den Vektoren in (1). Dort wird *works* als x_{b1} notiert und $x_{b2,b3}$ definieren die Umgebung des Wortes. Die Umgebung besteht sowohl aus der Folge der Wörter, für die bereits eine Wortart bestimmt worden ist, und deren Wortarten (vgl. x_{b2}) als auch aus der Folge der Wörter, für die die Wortart nach x_{b1} bestimmt wird (vgl. x_{b3}). Der Vektor y_b entspricht der Wortart/Kategorie zu x_{b1}.

$$x_b = \begin{pmatrix} \text{works} \\ (\text{Satya}, \text{NNP}, \text{Nadella}, \text{NNP}) \\ (\text{for}, \text{Microsoft}) \end{pmatrix} \quad y_b = \begin{pmatrix} \text{VBZ} \end{pmatrix} \tag{1}$$

Für die jeweiligen Wörter *Satya* und *Microsoft* sind die Tupel (x_a, y_a) und (x_e, y_e) in (2) und (3) dargestellt. Während der Trainingsphase des Supervised-Machine-Learning-Verfahrens wird jedes Wort innerhalb des Textkorpus in solch ein Tupel übertragen. Anschließend können basierend auf den Tupeln mathematische Analysen stattfinden. Generell findet dieser Schritt bei allen Werkzeugen der natürlichen Sprachverarbeitung statt – ebenfalls für das Parsing, das in Abschnitt 4 erläutert wird.

$$x_a = \begin{pmatrix} \text{Satya} \\ () \\ (\text{Nadella}, \text{works}, \text{for}, \text{Microsoft}) \end{pmatrix} \quad y_a = \begin{pmatrix} \text{NNP} \end{pmatrix} \tag{2}$$

$$x_e = \begin{pmatrix} \text{Microsoft} \\ (\text{Satya}, \text{NNP}, \text{Nadella}, \text{NNP}, \text{works}, \text{VBZ}, \text{for}, \text{IN}) \\ () \end{pmatrix} \quad y_e = \begin{pmatrix} \text{VBZ} \end{pmatrix} \tag{3}$$

Für die automatische Erkennung von Gesetzmäßigkeiten müssen Eingangsdaten x in einen Vektor $\hat{x} \in \mathbb{R}^n$ transformiert werden. Dazu wird eine Menge von Funktionen $f_i : X \to \mathbb{R}$, die x nach $\mathbb{R}$ übertragen, verwendet, die als Feature-Funktionen bezeichnet werden. Der Vektor $\hat{x}$ hat basierend auf den Feature-Funktionen folgenden Aufbau:

$$\hat{x} = \begin{pmatrix} f_1(x) \\ \vdots \\ f_n(x) \end{pmatrix}$$

[6] Reese [26, Seite 18] gibt eine Einführung zur Wortartbestimmung (engl. Part-of-Speech Tagging).

Bei der natürlichen Sprachverarbeitung werden Indikatorfunktionen verwendet, die angeben, ob ein Wort ein bestimmtes Kriterium erfüllt. Zum Beispiel werden u. a. folgende Indikatorfunktionen für die Wortartbestimmung eingesetzt (der Feature-Vektor $\hat{x}_b$ für (1) ist in (4) skizziert):

Wortendungen

$$f_i(x) = \begin{cases} 1 & x \text{ endet mit } s \\ 0 & \text{sonst} \end{cases}$$

Groß-/Kleinschreibung

$$f_j(x) = \begin{cases} 1 & x \text{ beginnt mit einem Großbuchstaben} \\ 0 & \text{sonst} \end{cases}$$

Bestimmte Wörter

$$f_k(x) = \begin{cases} 1 & x = \textit{the} \\ 0 & \text{sonst} \end{cases}$$

Umgebung von Wörtern

$$f_l(x) = \begin{cases} 1 & \text{Das vorangestellte Wort von } x \text{ entspricht } \textit{the} \\ 0 & \text{sonst} \end{cases}$$

$$f_m(x) = \begin{cases} 1 & \text{Die Wortart des vorangestellten Wortes von } x \text{ entspricht } \textit{NNP} \\ 0 & \text{sonst} \end{cases}$$

$$\hat{x}_b = \begin{pmatrix} \vdots \\ f_i(x_b) \\ f_j(x_b) \\ f_k(x_b) \\ f_l(x_b) \\ f_m(x_b) \\ \vdots \end{pmatrix} = \begin{pmatrix} \vdots \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ \vdots \end{pmatrix} \tag{4}$$

Basierend auf den Feature-Funktionen $f_i : X \to \mathbb{R}$ werden bei gängigen Supervised-Machine-Learning-Verfahren Wahrscheinlichkeitsfunktionen [40] ermittelt[7], um für unbekannte x eine entsprechende Klasse Y bestimmen zu können. Y entspricht somit einer Zufallsvariablen.

[7] Unter anderem verwenden Cutting u. a. [4], Ratnaparkhi [24], Sha und Pereira [28] und Klein und Manning [18] Wahrscheinlichkeitsfunktion.

Während der Bestimmung von Y für ein gegebenes x wird für alle aus den Trainingsdaten bekannten y die bedingte Wahrscheinlichkeit $p(y \mid \hat{x})$ bestimmt und y mit der höchsten bedingten Wahrscheinlichkeit entspricht der gesuchten Zufallsvariablen Y. Um letztendlich $p(Y \mid \hat{x})$ zu bestimmen, wird mit allen Tupeln (x, y) aus den Trainingsdaten ein Gleichungssystem aufgestellt. Die Form dieses Gleichungssystem hängt vom gewählten Supervised-Machine-Learning-Verfahren ab und liefert eine Lösung für $p(Y \mid \hat{x})$.

Durch diese Art der Notation und Anwendung ist es möglich, dass Werkzeuge der natürlichen Sprachverarbeitung den Kontext zu einem gewissen Grad auflösen können. Bei dem englischen Wort *flies* ist es durch eine isolierte Betrachtung nicht möglich zu sagen, ob es sich um das Verb *fliegen* oder um das Nomen *Fliegen* (die Insekten) handelt. Feature-Funktionen wie $f_l(x)$ oder $f_m(x)$ beziehen die Umgebung (den Kontext) mit ein. Folgt das Wort *flies* auf den Artikel *the*, ist mit hoher Wahrscheinlichkeit das Nomen gemeint (vgl. $f_l(x)$). Folgt das Wort *flies* auf ein Nomen (z. B. eine Person), ist mit einer gewissen Wahrscheinlichkeit das Verb gemeint (vgl. $f_m(x)$). Somit fließt der Kontext durch die Feature-Funktionen mit in die Bestimmung von $p(Y \mid \hat{x})$ ein.

3.1 Maximum-Entropie

Maximum-Entropie ist ein Supervised-Machine-Learning-Verfahren, bei dem eine Wahrscheinlichkeitsfunktion $p(Y \mid \hat{x})$ ermittelt wird. Diese Wahrscheinlichkeitsfunktion wird so gewählt, dass die Entropie der Zufallsvariablen maximal ist. Um diese Eigenschaft besser zu verstehen, wird im Folgenden der Begriff der Entropie beschrieben.

3.1.1 Entropie

In der Informationstheorie ist die Entropie ein Maß für den mittleren Informationsgehalt eines Zeichens eines Alphabets [29, 36]. Die Entropie basiert auf dem Informationsgehalt (5), der ein logarithmisches Maß ist und angibt, wie viel Informationen in einem Zeichen z enthalten sind. Zur Bestimmung des Informationsgehalts eines Zeichens z wird dessen Wahrscheinlichkeit $p(z)$ benötigt.

$$I(z) = \log_2 \left(\frac{1}{p(z)} \right) = -\log_2 \left(p(z) \right) \tag{5}$$

Das Maß $I(z)$ weist Zeichen mit hoher Wahrscheinlichkeit einen niedrigen Informationsgehalt zu, Zeichen mit geringer Wahrscheinlichkeit erhalten hingegen einen hohen Informationsgehalt. In der Datenkompression werden somit Zeichen mit geringem Informationsgehalt durch kurze Bitfolgen kodiert (weil diese Zeichen häufig auftreten) und Zeichen mit hohem Informationsgehalt durch lange Bitfolgen kodiert (weil diese Zeichen selten auftreten) – vgl. z. B. Huffman-Kodierung [15].

Die Entropie H_1 ist nach Shannon [29] als Erwartungswert des Informationsgehalts einer Zufallsvariablen Y aus einem Alphabet $\mathcal{Z}$ definiert (6).

$$H_1 = E\left[I(Y)\right] = \sum_{z \in \mathcal{Z}} p(z) I(z) = -\sum_{z \in \mathcal{Z}} p(z) \log_2 p(z) \tag{6}$$

In der Datenkompression beschreibt die Entropie die Güte des Kompressionsalgorithmus. Desto höher dieses Maß ist, desto weniger Bits werden durch ineffiziente Kodierung verschwendet. Das Optimum der Kodierung ist durch die maximale Entropie H_{max} definiert (7). H_1 entspricht H_{max} genau dann, wenn $p(z) = \frac{1}{|\mathcal{Z}|} \forall z \in \mathcal{Z}$, d. h alle Zeichen treten mit gleicher

Wahrscheinlichkeit auf. Kompressionsraten, wie beispielsweise durch Werkzeuge wie Winzip angezeigt werden, lassen sich durch $\frac{H_1}{H_{max}}$ bestimmen.

$$H_{max} = E\left[I(Y)\right] = \sum_{z \in \mathcal{Z}} \frac{1}{|Z|} I(z) = -\sum_{z \in \mathcal{Z}} \frac{1}{|Z|} \log_2 \frac{1}{|Z|} \tag{7}$$

3.1.2 Maximum-Entropie

Bei der Anwendung von Maximum-Entropie als Supervised-Machine-Learning-Verfahren, wird die Wahrscheinlichkeitsfunktion gesucht, die alle durch die Feature-Funktionen $f : X \to \mathbb{R}$ aufgestellten Bedingungen erfüllt und dessen Entropie maximal ist [16, 17, 10]. Es gibt eine Menge $\mathcal{P}$, die alle Wahrscheinlichkeitsfunktionen $p(Y \mid \hat{x})$ beinhaltet, die die Bedingungen aller Feature-Funktionen erfüllen. Die Wahrscheinlichkeitsfunktion (8), die die Bedingung erfüllt, dass die Entropie H_1 maximal ist, heißt $p^*(Y \mid \hat{x})$. Die Trainingsdatenmenge, die die Tupel (x, y) beinhaltet, wird mit $\mathcal{T}$ bezeichnet.

$$p^*(Y \mid \hat{x}) = \underset{p(Y \mid \hat{x}) \in \mathcal{P}}{\arg\max} \left(H_1\left(I(Y \mid \hat{x})\right)\right) = \underset{p(Y \mid \hat{x}) \in \mathcal{P}}{\arg\max} \left(- \sum_{(\hat{x}, y) \in \mathcal{T}} p(y|\hat{x}) \log_2 p(y|\hat{x}) \right) \tag{8}$$

Die Maximierung bzgl. der Entropie H_1 bewirkt, dass die Wahrscheinlichkeitsfunktion $p^*(Y \mid \hat{x})$ möglichst gleichverteilt ist – H_{max} entspricht Gleichverteilung. Durch diesen Ansatz wird sichergestellt, dass bei der Ermittlung von $p^*(Y \mid \hat{x})$ keine willkürlichen Einschränkungen getroffen werden, wenn notwendige Informationen fehlen. Nach Jaynes [16] und Good [10] ist dies die einzige Annahme, die getroffen werden kann, ohne einen systematischen Fehler zu machen.

Die möglichen Wahrscheinlichkeitsfunktionen aus $\mathcal{P}$ sind im Allgemeinen unbekannt. Durch die Trainingsdaten, die dem Maximum-Entropie-Verfahren vorgegeben sind, können die Wahrscheinlichkeitsfunktionen aus $\mathcal{P}$ durch empirische Daten angenähert werden. Mit den Feature-Funktionen kann der Erwartungswert $\tilde{E}_1$ von f_i bzgl. einer Klasse y angenähert werden (9). Dabei gibt $\tilde{p}(\hat{x}, y)$ die relative Häufigkeit von $(\hat{x}, y) \in \mathcal{T}$ an.

$$\tilde{E}_1(f_i, y) = \sum_{(\hat{x}, y) \in \mathcal{T}} \tilde{p}(\hat{x}, y) f_i(x) \tag{9}$$

Basierend auf dem Multiplikationssatz der bedingten Wahrscheinlichkeit (10) kann der Erwartungswert der Feature-Funktion f_i über die rechte Seite des Multiplikationssatz ebenfalls angenähert werden (11). Dabei entspricht $\tilde{p}(\hat{x})$ der empirischen Verteilungsfunktion von $\hat{x}$.

$$\tilde{p}(a, b) \approx \tilde{p}(a)\, p(b|a) \tag{10}$$

$$\tilde{E}_2(f_i, y) = \sum_{(\hat{x}, y) \in \mathcal{T}} \tilde{p}(\hat{x}) p(y|\hat{x}) f_i(\hat{x}) \tag{11}$$

Aufgrund der Bedingung, dass die Erwartungswerte $\tilde{E}_1$ und $\tilde{E}_2$ für alle f_i und alle $y \in \mathcal{T}$ gleich sind, stellt sich ein Gleichungssystem auf, dass Lösungen für $p(Y|\hat{x})$ liefert. Dieses Gleichungssystem lässt sich aber weiterhin durch alle $p(Y|\hat{x}) \in \mathcal{P}$ lösen. Durch die Bedingung, dass H_1 maximal sein soll, ist $p(Y|\hat{x})$ eindeutig bestimmt. Dieses Gleichungssystem kann mit Lagrange-Multiplikatoren [14] und Generalized-Iterative-Scaling [5] gelöst werden.

4 Parsing für natürliche Sprachen

Zum grundlegenden Verständnis und zur Einführung, wie die Struktur von natürlicher Sprache mittels Supervised-Machine-Learning-Verfahren ermittelt werden kann, wurde sich in den vorherigen Abschnitten auf die Wortartbestimmung beschränkt. Um den gesamten Ableitungsbaum eines Satzes zu ermitteln, ist die Wortartbestimmung notwendig (vgl. Abbildung 5). Die Wortartbestimmung kann mit anschließendem Parsing angewendet werden, sowie als eingebettetes Verfahren im Parsing statt finden. Der Parser von Adwait Ratnaparkhi arbeitet mit letzterem Ansatz. Dadurch ist nur die Segmentierung eines Textes in Sätze und Wörter als Vorverarbeitungsschritt notwendig.

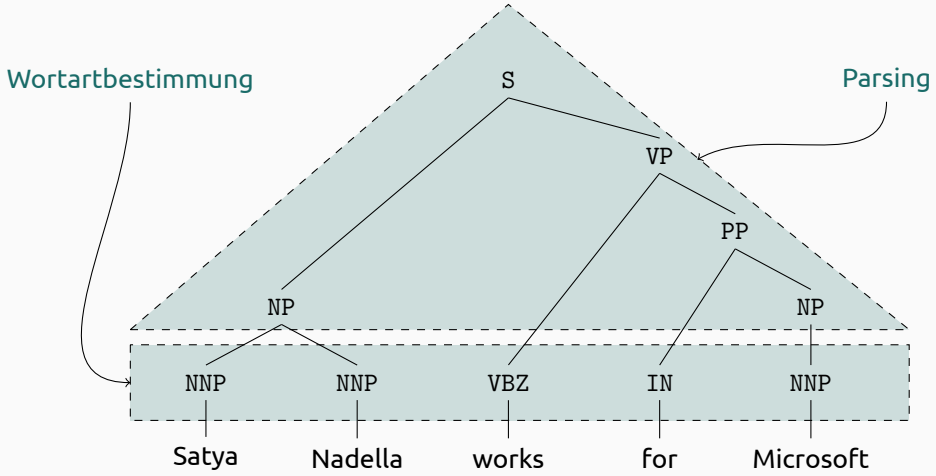


Abbildung 5: Parse-Tree der durch Parsing für den Satz *„Satya Nadella works for Microsoft"* ermittelt werden soll

Im folgenden wird nun anhand des Parser von Adwait Ratnaparkhi gezeigt, wie mittels Feature-Funktionen für Sätze Ableitungsbäume generiert werden (vgl. oberer Teil der Abbildung 5). Das Parsing-Verfahren von Adwait Ratnaparkhi setzt Maximum-Entropie als Supervised-Machine-Learning-Verfahren ein. Der Schritt der Wortartbestimmung kann in der Parser eingebettet werden, da die Wortbestimmung ebenfalls Maximum-Entropie einsetzt [24]. Die Features der Wortartbestimmung wurden in Abschnitt 3 durch $f_i, \ldots, f_m$ skizziert.

4.1 Parsing-Verfahren von Adwait Ratnaparkhi

In der Trainingsphase des Parser von Adwait Ratnaparkhi werden vier Wahrscheinlichkeitsfunktionen $p_O(Y_O|h)$ bestimmt, die dem Parser erlauben, einen Syntaxbaum schrittweise von unten nach oben aufzubauen – der Parser funktioniert ähnlich zu einem Buttom-up Parser für formale Sprachen und ebenfalls einen Vorverarbeitungsschritt ähnlich zur Tokenisierung eines Quelltextes benötigt.

Die vier Wahrscheinlichkeitsfunktionen $p_O(Y_O|h)$ werden jeweils von den vier Operationen TAG, CHUNK, BUILD und CHECK verwendet. Mit $\mathcal{T}$ wird hier der annotierte Textkorpus (auch Treebank genannt) bezeichnet, der den Trainingsdaten entspricht. Bevor die Operationen im einzelnen erklärt werden, wird die Wahrscheinlichkeitsfunktion $p_O(Y_O|h)$ genauer spezifiziert:

- $p_O(Y_O|h)$ ist eine bedingte Wahrscheinlichkeitsfunktion, d. h. Y_O wird unter der Bedingung h bestimmt (vgl. Downey [6], Georgii [9] und Wikipedia® [33]).
- Der Index O steht für die entsprechende Operation: $O \in \{TAG, CHUNK, BUILD, CHECK\}$
- Durch h ist das vorhandene und bisherige ermittelte Wissen während des Parsing-Vorgangs bezeichnet – h entspricht der Kodierung des bisher ermittelten partiellen Syntaxbaumes[8]. h wird auch als Kontext oder Historie bezeichnet: $h \in \{x_i \mid x_i \in \mathcal{T}' \supset \mathcal{T}\}$. h liegt in $\mathcal{T}'$, der Obermenge von $\mathcal{T}$, da auch Beobachtungen auftreten können, die nicht innerhalb der Treebank erfasst sind. Der Parser ist durch Maximum-Entropie in der Lage eine Wahrscheinlichkeit mit der Bedingung $h \in \{x_i \mid x_i \in \mathcal{T}'\}$ zu ermitteln.
- Y_O ist die Klasse, für die eine Wahrscheinlichkeit unter der Bedingung h bestimmt wird. In den Operationen werden für alle möglichen Klassen die Wahrscheinlichkeiten berechnet und die Klasse mit der höchsten Wahrscheinlichkeit wird für die jeweilige Operation gewählt.

Im Folgenden werden die vier Operationen TAG, CHUNK, BUILD und CHECK beschrieben. Die bedingte Wahrscheinlichkeitsfunktion $p_O(Y_O|h)$ wird mittels Maximum-Entropie bestimmt. Die Operationen werden der Reihe nach ausgeführt, wobei BUILD und CHECK im Wechsel ausgeführt werden, bis der Syntaxbaum für einen Satz generiert wurde (vgl. Abbildung 6).

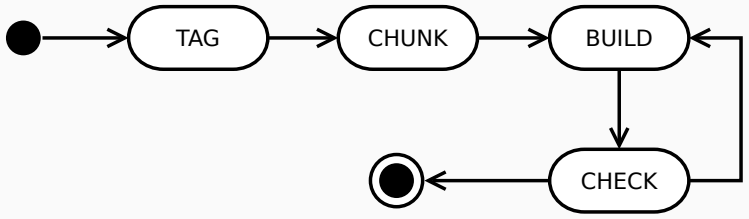


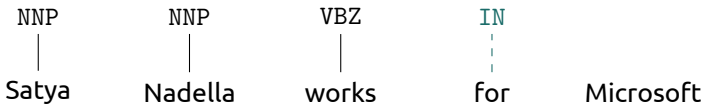
Abbildung 6: Aktivitätsdiagramm der Operationen des Parsers von Adwait Ratnaparkhi

Es wird in den folgenden Abschnitten jeweils beispielhaft spezifiziert, welche Werte h und Y_O annehmen können. Nach jeder Operation werden neue Elemente zu h hinzugefügt und diese mit Mintgrün gekennzeichnet.

TAG Diese Operation weist jedem Wort eines Satzes eine Wortart zu. Dabei wird die Wahrscheinlichkeitsfunktion $p_{TAG}(Y_{TAG}|h)$ verwendet.

- Y_{TAG} ist eine mögliche Wortart (engl. Part-of-Speech-Tag (POS-Tag)), die bestimmt werden soll. Die Menge der möglichen POS-Tags stammt aus der Treebank, $Y_{TAG} \in \{y_{TAG,i} \mid y_{TAG,i} \in \mathcal{T}\}$. Beispielsweise wird das Penn-Treebank-Tag-Set [21] in vielen Korpora verwendet.
- h entspricht den bisher ermittelten POS-Tags. Im folgenden Beispiel entspricht h der Folge von POS-Tags `NNP`, `NNP` und `VBZ` (vgl. POS-Tags in schwarz).

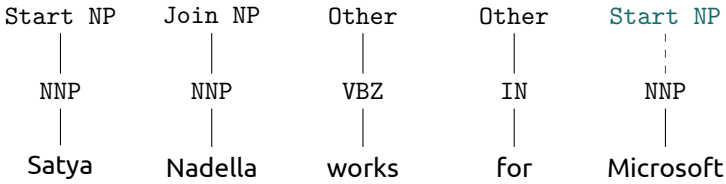
[8] Durch h wird eine andere aber inhaltlich ähnliche Schreibweise zu x_b aus Abschnitt 3 definiert.



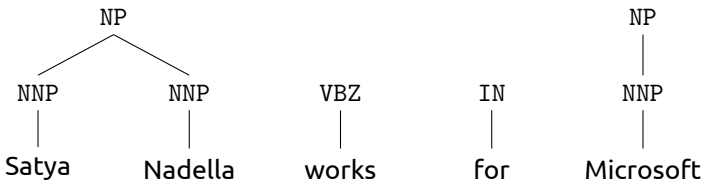
Es werden für alle möglichen Y_{TAG} die Wahrscheinlichkeit mittels $p_{TAG}(Y_{TAG}|h)$ für das Wort *„for"* bestimmt. In diesem Fall hat $p_{TAG}(\texttt{IN}|h)$ die höchste Wahrscheinlichkeit. Die Operation TAG wird für alle Tokens durchgeführt bis allen Tokens ein POS-Tag zugewiesen wurde[9].

CHUNK Diese Operation stellt für jedes Token fest, ob dieses Token mit umliegenden Tokens zu einer Phrase/Konstituente zusammengefasst werden kann. Dadurch entstehen erste Teilbäume des finalen Syntaxbaumes.

- Y_{CHUNK} entspricht einer von drei verschiedenen Klassentypen: `Start X`, `Join X` oder `Other`, `X` steht dabei für ein mögliches Phrase-Tag (z. B. entsprechende Penn-Treebank-Phrase-Tags [21]). `Start X` fasst das aktuelle Token mit allen folgenden Tokens zusammen, die mit der entsprechenden `Join X` Klasse annotiert sind. `Other` fasst die entsprechenden Token nicht zusammen.
- h entspricht den ermittelten POS-Tags und den bisher ermittelten CHUNK-Klassen Y_{CHUNK}. Im folgenden Beispiel entspricht $h = \ldots$ `IN`, `NNP`, `Start NP`, `Join NP`, `Other`, `Other` (vgl. schwarze CHUNK-Klassen).



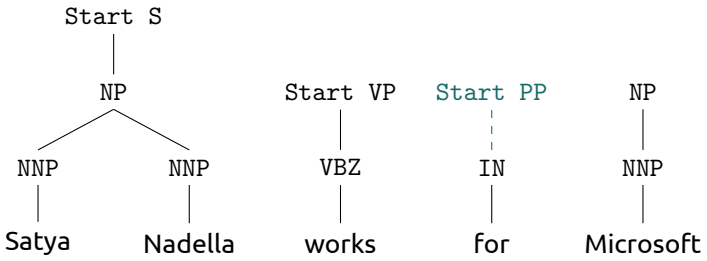
Es werden für alle möglichen Y_{CHUNK} die Wahrscheinlichkeit mittels $p_{CHUNK}(Y_{CHUNK}|h)$ für das Wort *„Microsoft"* bestimmt. In diesem Fall hat $p_{CHUNK}(\texttt{Start NP}|h)$ die höchste Wahrscheinlichkeit. Die Operation CHUNK wird für alle Tokens durchgeführt. Die vorherige Darstellung entspricht dem folgenden partiellen Syntaxbaum.



BUILD Die BUILD-Operation entscheidet, ob eine neue Phrase/Konstituente gebildet werden soll oder ob eine Phrase/Konstituente mit der unmittelbar davorliegenden Phrase/Konstituente zusammengefasst werden soll. Die BUILD-Operation benötigt einen partiellen Syntaxbaum, der erste Konstituenten enthält, weswegen die CHUNK-Operation der BUILD-Operation vorangestellt ist.

[9] Dies geschieht nach dem gleichen Verfahren, wie es Ratnaparkhi [24] beschreibt.

- Y_{BUILD} entspricht einer von zwei Klassentypen: `Start X` oder `Join X`. `X` steht dabei für ein mögliches Phrase-Tag (z. B. entsprechende Penn-Treebank-Phrase-Tags [21]).
- h entspricht den ermittelten POS-Tags, den ermittelten CHUNK-Klassen und den bisher ermittelten BUILD-Klassen sowie den bisher ermittelten CHECK-Klassen. Im folgenden Beispiel entspricht $h = \ldots$ `Other`, `Other`, `Start NP`, `Start S`, `no`, `Start VP`, `no` (vgl. schwarze BUILD-Klassen). Die CHECK-Operation hat in diesem Beispiel immer die Klasse `no` als Ergebnis geliefert, dessen Bedeutung mit der nächsten Operation erklärt wird.



Es werden für alle möglichen Y_{BUILD} die Wahrscheinlichkeit mittels $p_{BUILD}(\ Y_{BUILD}\ |h)$ für den Teilbaum oberhalb *„for"* bestimmt – Y_{BUILD} wird immer für den am weitesten links-stehenden Teilbaum ohne BUILD-Klasse ermittelt. In diesem Fall hat $p_{BUILD}(\texttt{Start PP}|h)$ die höchste Wahrscheinlichkeit. Im Anschluss wird nun die CHECK-Operation ausgeführt, auf die dann wiederum die BUILD-Operation ausgeführt wird, bis der Syntaxbaum vollständig ist.

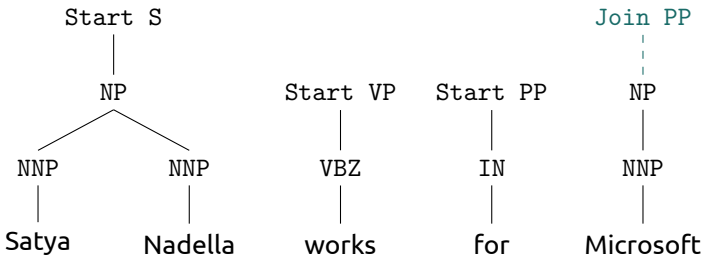
CHECK Die CHECK-Operation entscheidet, ob die zuletzt gebildete Phrase/Konstituente abgeschlossen wird, d. h. zu einem Teilbaum zusammengefasst wird.

- Y_{CHECK} entspricht entweder der Klasse `no` oder `yes`. `yes` bedeutet, dass die zuletzt gebildete Konstituente zu einem Teilbaum vereinigt wird.
- h wird aus der letzten BUILD-Operation fortgeführt.

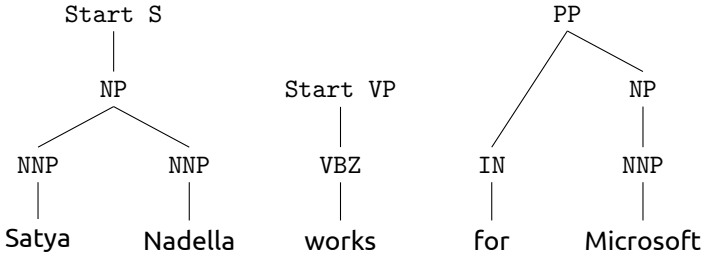
Es werden für alle möglichen Y_{CHECK} die Wahrscheinlichkeit mittels $p_{CHECK}(\ Y_{CHECK}\ |h)$ für den Teilbaum oberhalb *„for"* bestimmt. In diesem Fall hat $p_{CHECK}(\texttt{no}|h)$ die höchste Wahrscheinlichkeit. Der partielle Syntaxbaum bleibt unverändert.

Die Ausgabe `yes` der CHECK-Operation entspricht somit dem Reduce-Schritt eines Bottom-up-Parsers. Im Folgenden wird das Zusammenspiel der BUILD- und CHECK-Operation verdeutlicht, indem schrittweise gezeigt wird, wie die beiden Operationen den Syntaxbaum vervollständigen.

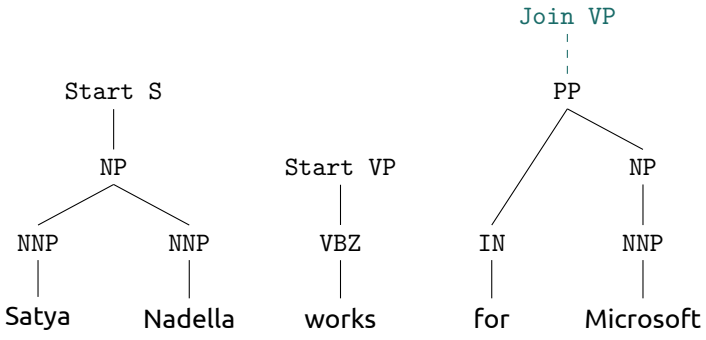
1. Anwendung der BUILD-Operation auf den am weitesten links-stehenden Teilbaum ohne BUILD-Klasse (Teilbaum überhalb *„Microsoft"*). $p_{BUILD}(\texttt{Join PP}|h)$ hat die höchste Wahrscheinlichkeit und `Join PP` wird neues Element von h.



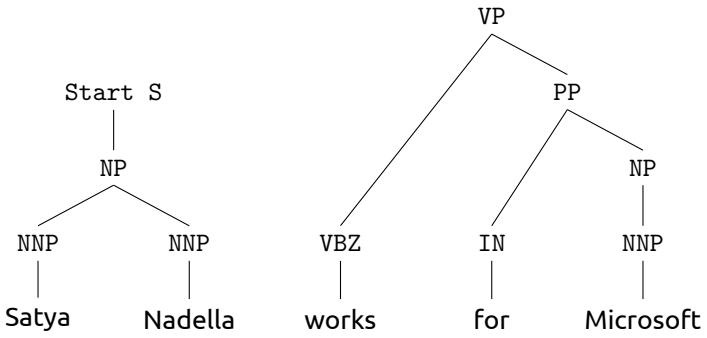
2. Anwendung der CHECK-Operation auf die zuletzt gebildete Konstituente (zusammengesetzt aus `Start PP` und `Join PP`). $p_{CHECK}(\texttt{yes}|h)$ hat die höchste Wahrscheinlichkeit und `yes` wird neues Element von h. Teilbäume werden zu einer Phrase zusammengefasst.



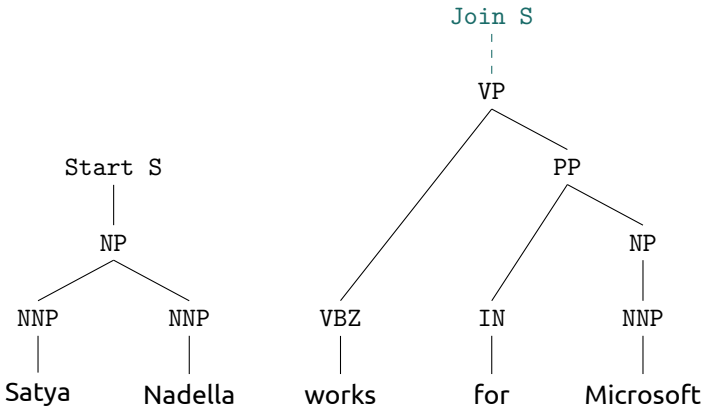
3. Anwendung der BUILD-Operation auf den am weitesten links-stehenden Teilbaum ohne BUILD-Klasse (Teilbaum überhalb „*for Microsoft*"). $p_{BUILD}(\texttt{Join VP}|h)$ hat die höchste Wahrscheinlichkeit und `Join VP` wird neues Element von h.



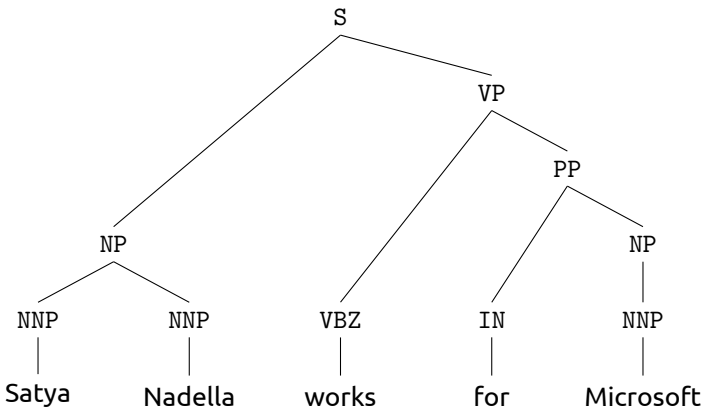
4. Anwendung der CHECK-Operation auf die zuletzt gebildete Konstituente (zusammengesetzt aus `Start VP` und `Join VP`). $p_{CHECK}(\texttt{yes}|h)$ hat die höchste Wahrscheinlichkeit und `yes` wird neues Element von h. Teilbäume werden zu einer Phrase zusammengefasst.



5. Anwendung der BUILD-Operation auf den am weitesten links-stehenden Teilbaum ohne BUILD-Klasse (Teilbaum überhalb „*works for Microsoft*"). $p_{BUILD}(\texttt{Join S}|h)$ hat die höchste Wahrscheinlichkeit und `Join S` wird neues Element von h.



6. Anwendung der CHECK-Operation auf die zuletzt gebildete Konstituente (zusammengesetzt aus `Start S` und `Join S`). $p_{CHECK}(\texttt{yes}|h)$ hat die höchste Wahrscheinlichkeit und `yes` wird neues Element von h. Teilbäume werden zu einer Phrase vereint. Der Syntaxbaum für den Satz aus Abbildung 2 ist vollständig gebildet worden.



Wie oben gezeigt wurde, ist der Parser von Adwait Ratnaparkhi mit den Wahrscheinlichkeitsfunktionen $p_O(Y_O|h)$ in der Lage sukzessive Syntaxbäume für Sätze der natürlichen Sprache mit einer gewissen Wahrscheinlichkeit zu bilden.

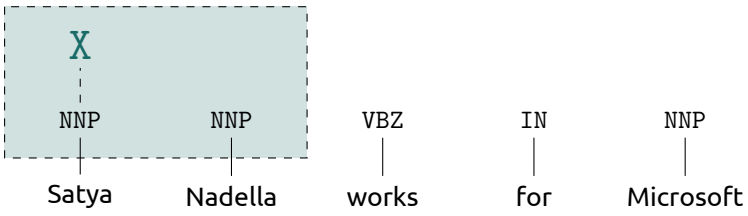
4.2 Feature-Funktionen des Parser von Adwait Ratnaparkhi

Damit über Maximum-Entropie die Wahrscheinlichkeitsfunktionen p_O^* ermittelt werden können, werden Feature-Funktionen $f_{i,O}(h)$ basierend auf der Historie h benötigt. Im Folgenden werden für die Operationen des Parsers von Adwait Ratnaparkhi beispielhaft Feature-Funktionen beschrieben. Zur weiteren Verdeutlichung wird ebenfalls ein partieller Ableitungsbaum angegeben, der anzeigt, ob die Feature-Funktion den Wert eins oder null annimmt. Mit einem großen `X` wird angezeigt, an welcher Stelle p_O^* angewendet würde und mit einem Rechteck wird gezeigt, welchen Teil des partiellen Ableitungsbaum die Feature-Funktion auswertet. Feature-Funktionen für die TAG-Operation werden hier ausgelassen, da Beispiele bereits in Abschnitt 3 skizziert worden sind – die vollständige Liste für die TAG-Operation ist von Ratnaparkhi [24] beschrieben.

CHUNK

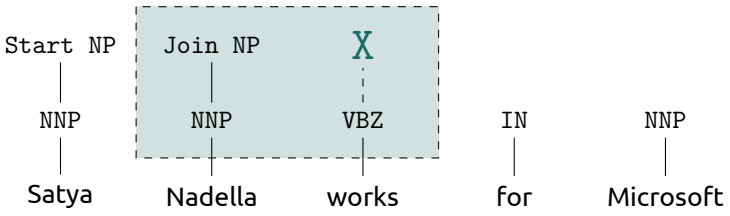
$$f_{i,CHUNK}(h) = \begin{cases} 1 & \text{Wortart des aktuellen und daraufliegenden Wortes ist } \texttt{NNP} \\ 0 & \text{sonst} \end{cases}$$

Im folgenden partiellen Ableitungsbaum wird `Start NP` bei der Anwendung von p^*_{CHUNK} erwartet. Die Feature-Funktion $f_{i,CHUNK}(h)$ liefert hier den Wert eins.



$$f_{j,CHUNK}(h) = \begin{cases} 1 & \text{Wortart des aktuellen Wortes ist } \texttt{VBZ} \text{ und Wortart des} \\ & \text{vorangestellten Wortes ist } \texttt{NNP} \text{ mit CHUNK-Tag } \texttt{Join NP} \\ 0 & \text{sonst} \end{cases}$$

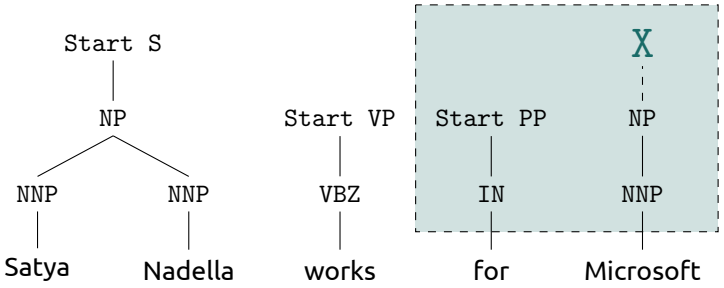
Im folgenden partiellen Ableitungsbaum wird `Other` bei der Anwendung von p^*_{CHUNK} erwartet. Die Feature-Funktion $f_{j,CHUNK}(h)$ liefert hier den Wert eins.



BUILD

$$f_{i,BUILD}(h) = \begin{cases} 1 & \text{Der vorangestellte Teilbaum ist mit } \texttt{Start PP} \text{ annotiert und der} \\ & \text{aktuelle Teilbaum ist mit } \texttt{NP} \text{ annotiert.} \\ 0 & \text{sonst} \end{cases}$$

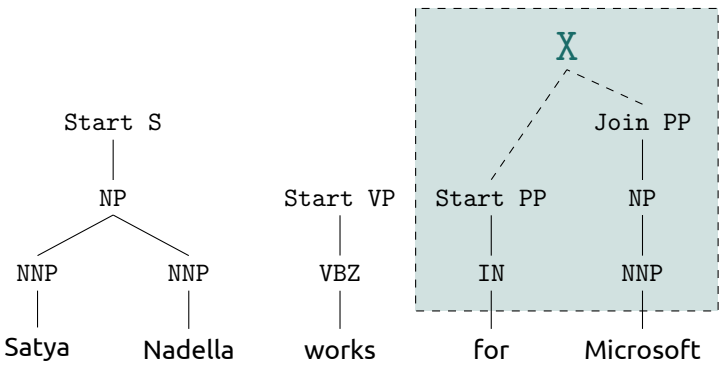
Im folgenden partiellen Ableitungsbaum wird `Join PP` bei der Anwendung von p^*_{BUILD} erwartet. Die Feature-Funktion $f_{i,BUILD}(h)$ liefert hier den Wert eins.



CHECK

$$f_{i,CHECK}(h) = \begin{cases} 1 & \text{Der vorangestellte Teilbaum ist mit } \texttt{Start PP} \text{ annotiert und für} \\ & \text{den aktuellen Teilbaum wurde } \texttt{Join PP} \text{ vorgeschlagen.} \\ 0 & \text{sonst} \end{cases}$$

Im folgenden partiellen Ableitungsbaum wird `yes` bei der Anwendung von p^*_{CHECK} erwartet. Die Feature-Funktion $f_{i,CHECK}(h)$ liefert hier den Wert eins.



Die dargestellten Feature-Funktionen werden für den Parser von Adwait Ratnaparkhi mittels verschiedener Vorlagefunktionen generiert[10]. Diese Vorlagefunktionen liefern basierend von der aktuellen Position (vgl. `X` in vorherigen partiellen Ableitungsbäumen) z. B. die Wortart des $n$ten Wortes im Teilbaum oder die Konstituente eines Teilbaumes. Diese werden auf alle

[10] Ratnaparkhi [25] beschreibt die vollständige Liste der Vorlagefunktionen.

möglichen Historien h in den Trainingsdaten angewendet und dadurch entstehen die Feature-Funktionen, die hier auszugsweise und beispielhaft skizziert worden sind. Mit dieser Menge von generierten Feature-Funktionen wird dann das Gleichungssystem, wie in Abschnitt 3.1 beschrieben, aufgestellt und die Wahrscheinlichkeitsfunktionen p_O^* können ermittelt werden.

4.3 Parsing-Qualität

Die Qualität des Parsing hängt maßgeblich von den Trainingsdaten ab. Diese sollten eine repräsentative Teilmenge der zu verarbeitenden natürlichen Sprache sein. Diese Teilmenge sollte die Eigenschaften der Sprache widerspiegeln, damit diese durch das Supervised-Machine-Learning-Verfahren erfasst werden und bei der Anwendung der ermittelten Wahrscheinlichkeitsfunktion erkannt werden.

Beim Parsen formaler Sprachen ist das Ergebnis durch die gegebene formale Grammatik definiert, d. h. wenn die gegebene Grammatik eindeutig ist, dann ist der Ableitungsbaum eindeutig. Somit besitzt das Parsing für formale Sprachen eine hundertprozentige Genauigkeit. Der Supervised-Machine-Learning-Ansatz besitzt diese Genauigkeit nicht, d. h. es können Ableitungsbäume nur mit einer gewissen Qualität (Zusagesicherheit) bestimmt werden.

In der Datenkompression kann die Qualität eines Kompressionsalgorithmus durch $\frac{H_1}{H_{max}}$ ermittelt werden. Dieses Maß kann für das Supervised-Machine-Learning-Verfahren Maximum-Entropie zur Qualitätsbestimmung nicht angewendet werden, da H_{max} keine Aussage über den ermittelten Ableitungsbaum trifft. Um eine Aussage über die Qualität des Parsings zu treffen, muss dieses Verfahren getestet werden.

Eine Technik zum Testen ist das Kreuzvalidierungsverfahren [38]. Dabei wird die Trainingsmenge in mindestens zwei Teilmengen unterteilt. Dann wird eine Menge als Testmenge gewählt und die Vereinigung der verbliebenen Mengen bildet die Trainingsmenge. Diese wird dann verwendet, um ein p^* zu ermitteln, um p^* auf der Testmenge anzuwenden. Die Ergebnisse können mit den Annotationen der Testmenge verglichen werden, um die Genauigkeit zu bestimmen. Dies wird mit allen Teilmengen zum Testen wiederholt. Der Durchschnitt der ermittelten Genauigkeiten stellt eine Annäherung für die Qualität des Modells an.

Bei den üblichen Parsing-Verfahren wird eine Genauigkeit von ca. 80% erreicht. Das Parsing-Verfahren von Adwait Ratnaparkhi erreicht ca. 87%. Detailtiere Informationen zur Genauigkeit des Parsers werden von Ratnaparkhi [25] beschrieben.

5 Zusammenfassung

Für die Verarbeitung von natürlicher Sprache ist ein wichtiger Zwischenschritt das Parsing, bei dem für Sätze der natürlichen Sprache Ableitungsbäume bestimmt werden. Dieses Verfahren ist vergleichbar zum Parsen formaler Sprachen, wie z. B. das Parsen eines Quelltextes. Die Parsing-Methoden der formalen Sprachen, z. B. Bottom-up-Parser, können nicht auf das Parsen der natürlichen Sprache übertragen werden, da keine Formalisierung der natürlichen Sprachen existiert [3, 12, 23, 30].

In den ersten Programmen, die natürliche Sprache verarbeiten [32, 41], wurde versucht die natürliche Sprache mit festen Regelmengen zu verarbeiten. Dieser Ansatz stieß jedoch schnell an seine Grenzen, da die Regelmenge nicht vollständig sowie nicht minimal ist und wegen der benötigten Menge an Regeln schwer zu verwalten ist. Die Korpuslinguistik [22] bot die Möglichkeit, die Regelmenge durch Supervised-Machine-Learning-Verfahren [2] abzulösen.

Teil der Korpuslinguistik ist es, große Textkorpora zu erstellen und diese mit sprachlichen Strukturen zu annotieren. Zu diesen Strukturen gehören sowohl die Wortarten als auch die Ableitungsbäume der Sätze. Vorteil dieser Methodik ist es, dass repräsentative Daten zur Verfügung stehen. Diese Daten werden genutzt, um mit Supervised-Machine-Learning-Verfahren die Gesetzmäßigkeiten der natürliche Sprachen zu erlernen.

Das Maximum-Entropie-Verfahren ist ein Supervised-Machine-Learning-Verfahren, das genutzt wird, um natürliche Sprache zu erlernen. Ratnaparkhi [25] nutzt Maximum-Entropie, um Ableitungsbäume für Sätze der natürlichen Sprache zu erlernen. Dieses Verfahren macht es möglich, die natürliche Sprache (abgebildet als Σ^*) trotz einer fehlenden formalen Grammatik zu parsen.

Literatur

[1] Alfred V. Aho u. a. *Compilers: Principles, Techniques, and Tools*. Second Edition. Pearson Education, Inc, 2006. ISBN: 0-321-48681-1.

[2] Jason Bell. *Machine Learning. Hands-On for Developers and Technical Professionals*. 2014. ISBN: 978-1-118-88906-0.

[3] Noam Chomsky. „Three Models for the Description of Language“. In: *Information Theory, IRE Transactions on* 2.3 (1956), S. 113–124. DOI: 10.1109/TIT.1956.1056813.

[4] Doug Cutting u. a. „A Practical Part-of-speech Tagger“. In: *Proceedings of the Third Conference on Applied Natural Language Processing*. ANLC '92. Trento, Italy: Association for Computational Linguistics, 1992, S. 133–140. DOI: 10.3115/974499.974523.

[5] J. N. Darroch und D. Ratcliff. „Generalized Iterative Scaling for Log-Linear Models“. In: *Ann. Math. Statist.* 43.5 (Okt. 1972), S. 1470–1480. DOI: 10.1214/aoms/1177692379.

[6] Allen B. Downey. *Think Stats*. 1. Aufl. O'Reilly Media, 2011. ISBN: 978-1-449-30711-0.

[7] Moritz Eysholdt und Heiko Behrens. „Xtext: Implement Your Language Faster Than the Quick and Dirty Way“. In: *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*. OOPSLA '10. ACM, 2010, S. 307–309. ISBN: 978-1-4503-0240-1. DOI: 10.1145/1869542.1869625.

[8] Norbert E. Fuchs, Kaarel Kaljurand und Gerold Schneider. „Attempto Controlled English Meets the Challenges of Knowledge Representation, Reasoning, Interoperability and User Interfaces“. In: *FLAIRS Conference*. Hrsg. von Geoff Sutcliffe und Randy Goebel. AAAI Press, 2006, S. 664–669.

[9] Hans-Otto Georgii. *Stochastik. Einführung in die Wahrscheinlichkeitstheorie und Statistik*. Walter de Gruyter GmbH, Jan. 2009. ISBN: 9783110215274. DOI: 10.1515/9783110215274.

[10] I. J. Good. „Maximum Entropy for Hypothesis Formulation, Especially for Multidimensional Contingency Tables“. In: *The Annals of Mathematical Statistics* 34.3 (1963), S. 911–934. ISSN: 0003-4851.

[11] Barbara Greene und Gerald Rubin. *Automatic Grammatical Tagging of English*. Department of Linguistics, Brown University, 1971.

[12] James Higginbotham. „English Is Not a Context-Free Language“. In: *Linguistic Inquiry* 15.2 (1984), S. 225–234. ISSN: 00243892, 15309150. URL: http://www.jstor.org/stable/4178381.

[13] John E. Hopcroft und Jeffrey D. Ullman. *Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie*. 3. Auflage. 1994. ISBN: 978-3486243659.

[14] Reiner Horst. *Nichtlineare Optimierung*. Bd. 1. Carl Hanser Verlag, 1979. ISBN: 978-3-446-12550-6.

[15] David A Huffman u. a. „A method for the construction of minimum-redundancy codes“. In: *Proceedings of the IRE* 40.9 (1952), S. 1098–1101.

[16] E. T. Jaynes. „Information Theory and Statistical Mechanics“. In: *Physical Review* 106.4 (Mai 1957), S. 620–630. ISSN: 0031-899X. DOI: 10.1103/physrev.106.620.

[17] E. T. Jaynes. „Information Theory and Statistical Mechanics. II“. In: *Physical Review* 108.2 (Okt. 1957), S. 171–190. ISSN: 0031-899X. DOI: 10.1103/physrev.108.171.

[18] Dan Klein und Christopher D. Manning. „Accurate Unlexicalized Parsing“. In: *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*. 2003, S. 423–430.

[19] Henry Kučera. *Computational Analysis of Present Day American English*. Brown University Press, 1967. ISBN: 978-0870571053.

[20] Christopher D. Manning. „Part-of-Speech Tagging from 97% to 100%: Is It Time for Some Linguistics?“ In: *Lecture Notes in Computer Science* (2011), S. 171–189. ISSN: 1611-3349. DOI: 10.1007/978-3-642-19400-9_14.

[21] Mitchell P. Marcus, Mary Ann Marcinkiewicz und Beatrice Santorini. „Building a large annotated corpus of English: The Penn Treebank“. In: *Computational Linguistics* 19.2 (1993), S. 313–330.

[22] Tony McEnery und Andrew Wilson. *Corpus Linguistics*. Edinburgh University Press, 2001. ISBN: 9780748611652.

[23] Geoffrey K. Pullum. „On Two Recent Attempts to Show That English is Not a CFL“. In: *Computational Linguistics* 10.3–4 (Juli 1984), S. 182–186. ISSN: 0891-2017.

[24] Adwait Ratnaparkhi. „A Maximum Entropy Model for Part-Of-Speech Tagging“. In: *Proceedings of the Empirical Methods in Natural Language Processing Conference (EMNLP)*. 1996, S. 133–142.

[25] Adwait Ratnaparkhi. „Learning to Parse Natural Language with Maximum Entropy Models“. In: *Machine Learning* 34 (1999), S. 151–175. ISSN: 0885-6125. DOI: 10.1023/A:1007502103375.

[26] Richard M. Reese. *Natural Language Processing with Java*. PACKT Publishing, 2015. ISBN: 978-1-78439-179-9.

[27] Uwe Schöning. *Theoretische Informatik - kurz gefasst*. 5. Auflage. Spektrum Akademischer Verlag, 2008. ISBN: 978-3-827-41824-1.

[28] Fei Sha und Fernando Pereira. „Shallow Parsing with Conditional Random Fields“. In: *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*. NAACL '03. Edmonton, Canada: Association for Computational Linguistics, 2003, S. 134–141. DOI: 10.3115/1073445.1073473.

[29] Claude Elwood Shannon. „A mathematical theory of communication“. In: *ACM SIGMOBILE Mobile Computing and Communications Review* 5.1 (2001), S. 3–55. ISSN: 1559-1662. DOI: 10.1145/584091.584093.

[30] Stuart Shieber. „Evidence Against the Context-Freeness of Natural Language“. In: *Linguistics and Philosophy* 8 (1985), S. 333–343. DOI: 10.1007/BF00630917.

[31] Prof. Dr. Klaus W. Wagner. *Theoretische Informatik. Eine kompakte Einführung*. Springer Verlag, 2013. ISBN: 978-3-540-01313-6. DOI: 10.1007/978-3-642-55452-0.

[32] Joseph Weizenbaum. „ELIZA—A Computer Program For the Study of Natural Language Communication Between Man And Machine“. In: *Communications of the ACM* 9.1 (1966), S. 36–45. ISSN: 0001-0782. DOI: 10.1145/365153.365168.

[33] Wikipedia®. *Bedingte Wahrscheinlichkeit*. 2016. URL: https://de.wikipedia.org/wiki/Bedingte_Wahrscheinlichkeit (besucht am 22. 04. 2016).

[34] Wikipedia®. *DBpedia*. 2016. URL: https://de.wikipedia.org/wiki/DBpedia (besucht am 22. 06. 2016).

[35] Wikipedia®. *ELIZA*. 2016. URL: https://de.wikipedia.org/wiki/ELIZA (besucht am 22. 06. 2016).

[36] Wikipedia®. *Entropie (Informationstheorie)*. 2016. URL: https://de.wikipedia.org/wiki/Entropie_%28Informationstheorie%29 (besucht am 22. 01. 2016).

[37] Wikipedia®. *Knowledge Graph*. 2016. URL: https://en.wikipedia.org/wiki/Knowledge_Graph (besucht am 22. 01. 2016).

[38] Wikipedia®. *Kreuzvalidierungsverfahren*. 2016. URL: https://de.wikipedia.org/wiki/Kreuzvalidierungsverfahren (besucht am 01. 07. 2016).

[39] Wikipedia®. *Lineare Regression*. 2016. URL: https://de.wikipedia.org/wiki/Lineare_Regression (besucht am 28. 06. 2016).

[40] Wikipedia®. *Wahrscheinlichkeitsfunktion*. 2016. URL: https://de.wikipedia.org/wiki/Wahrscheinlichkeitsfunktion (besucht am 28. 06. 2016).

[41] Terry Winograd. „Procedures as a Representation for Data in a Computer Program for Understanding Natural Language“. Diss. MIT, Feb. 1971.