Institut für Rechnergestützte Ingenieursysteme

Fakultät Informatik, Elektrotechnik und Informationstechnik

Universität Stuttgart
Universitätsstraße 38
D - 70569 Stuttgart

Diplomarbeit Nr. 3397

Internetgestützte Textanalyse zur Extraktion von Produktentwicklungswissen mittels OntoUSP: eine Machbarkeitsanalyse

Chen Wang

Studiengang:	INFORMATIK
Prüfer:	Univ-Prof. Hon-Prof. Dr. Dieter Roller
Betreuer:	M. Sc. Julian Eichhoff
begonnen am:	06.11.2012
beendet am:	08.05.2013
CR-Klassifikation:	I.2.7 I.5.2 J.6

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

Die Extraktion von Produkten, Produkteigenschaften und Werten der Produkteigenschaften aus einem natürlichen Text kann als ein Problem von Text Mining angesehen werden. Text Mining ist eine neue Disziplin, die in den letzten zehn Jahren entstanden ist, und es gibt keine generell akzeptierte Definition für Text Mining. Die folgende enge Definition wird für diese Ausarbeitung übernommen: „Mit dem Terminus Text Mining werden computergestützte Verfahren für die semantische Analyse von Texten bezeichnet, welche die automatische bzw. semi-automatische Strukturierung von Texten, insbesondere sehr großen Mengen von Texten, unterstützen.“ [1] Die natürliche Sprache besitzt keine Struktur. Aber es existieren Regeln, mit deren Hilfe aus den Wörtern die Phrasen und aus den Phrasen die Sätze aufgebaut werden. Um die natürlichen Texte zu bearbeiten, sollten die s.g. Regeln und Wortbedeutungen als Vorkenntnisse für Text Mining vorhanden sein. NLP bietet eine gute Möglichkeit für die Vorbearbeitung der natürlichen Sprache. Der natürliche Text wird durch syntaktische und semantische Analyse verarbeitet. Danach wird die Cluster Analyse durchgeführt und die Daten aus dem Clustering werden durch Filter gefiltert.

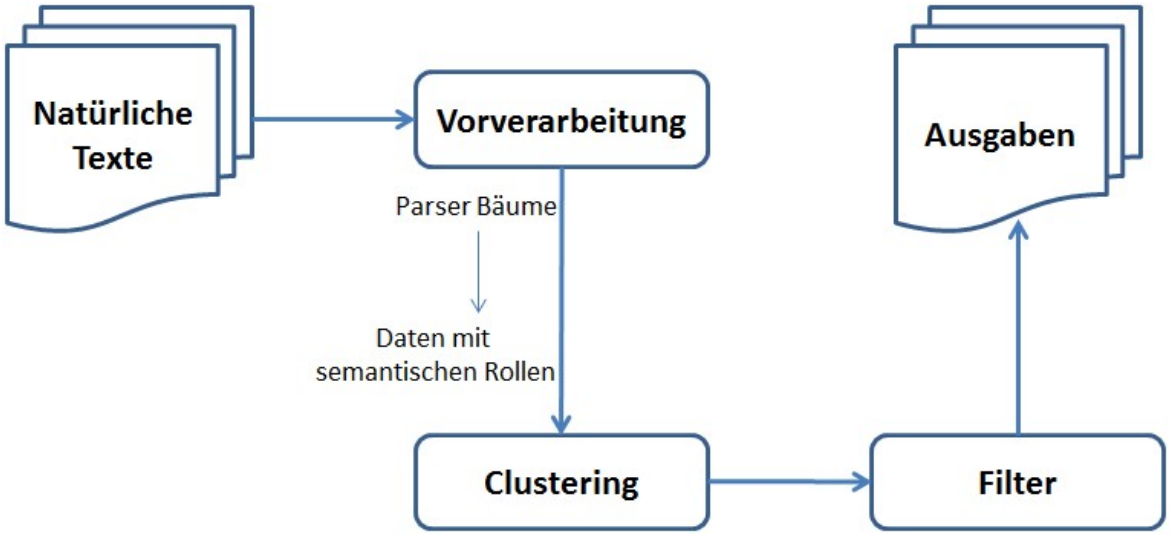


Abbildung 1: Der Ablauf von Data Mining

„Verarbeitung natürlicher Sprache (NLP: Natural language processing) ist ein Oberbegriff für alle Forschungs- und Anwendungsbereiche der Disziplinen Computerlinguistik (Computational Linguistik (CL)), der linguistischen Datenverarbeitung, der sprachorientierten Künstlichen-Intelligenz-Forschung und Sprachtechnologie.“ [2] Durch die syntaktische Analyse wird der natürliche Text in die formalen Ausdrücke umgewandelt, z.B. Prädikatlogik erster Stufe und Dependenzbäume in ein USP System. Aber die durch syntaktische Analyse erzeugten Strukturen sind mehrdeutig. Mit Hilfe von semantischer Analyse werden den Wörtern die

semantischen Rollen zugewiesen. „Semantische Rollen können zu der automatischen Textinhaltserschließung eines Dokuments beitragen, da sie eine formalisierte Repräsentation von Informationen zu den Sachverhalten und Ereignissen, die aus einem Text extrahiert werden können, sowie zu den Relationen zwischen den involvierten Entitäten darstellen.“ [3] Damit werden die Ambiguitäten und syntaktischen Variationen abgezogen, weshalb die semantische Analyse eine wichtige Rolle bei der Sprachanalyse spielt und die semantische Analyse auch eine Schwierigkeit für die Sprachanalyse darstellt. Wie in Abbildung 2 gezeigt, besteht die Sprachanalyse aus syntaktischer Analyse und semantischer Analyse und liefert die semantischen Repräsentationen für den natürlichen Text. Je besser diese semantische Repräsentation ist, desto exakter sind die extrahierten Informationen.

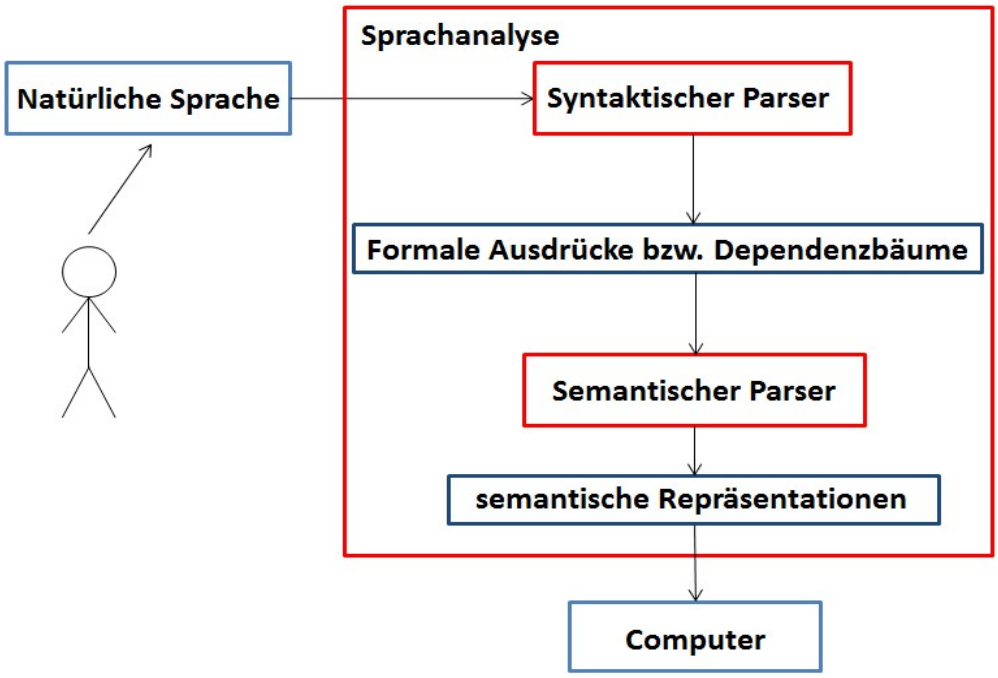


Abbildung 2 : Ablauf der Sprachanalyse

Vor mehreren Jahren wurden einige Ansätze für die semantische Analyse entwickelt. Zum Beispiel die Ansätze aus [4] und [5]. In [4] und [5] müssen einige semantische Parser als Muster manuell vordefiniert und die logischen Formen für jeden Satz angegeben werden, danach werden alle semantischen Parser mithilfe des Maschinenlernens auf den Text abgebildet. Dies ist offensichtlich sehr aufwendig. „Unsupervised Semantic Parsing“ (Abk. USP) ist der erste nicht überwachte Ansatz für semantische Analysen. Dieser Ansatz wurde von Hoifung Poon und Pedro Domingos im Jahr 2009 entwickelt und ermöglicht es mit der Hilfe von Markov Logik Netzwerk, eine vollautomatische semantische Analyse zu realisieren. „Für die Extraktion der Informationen beschränkt USP System sich darauf, wievielmal die gesuchte Information im Korpus angegeben wird. Wenn die Information selten im Korpus angegeben würde, wäre es schwierig zu extrahieren.“ [6] Des Weiteren wurde basierend auf „Unsupervised Semantic Parsing“ im Jahr 2010 “Ontology Unsupervised Semantic Parsing“ (Abk. OntoUSP) vorgeschlagen. „Während der Umwandlung der Sätze in logische Formen wird ISA Hierarchie

von Lambda-Formen durch die Clusteranalyse der logischen Ausdrücke generiert. Damit ist es möglich, eine Ontologie zu erstellen.“ [7] Die Informationen können trotzdem extrahiert werden, wenn die Informationen selten in dem Korpus vorkommen, weil die Informationen schon in der Ontologie eingesetzt werden.

1.1 Ziel der Diplomarbeit

In dieser Diplomarbeit wird geprüft, ob OntoUSP eine Methode ist, mit der das Produkt, die Produkteigenschaft und die Werte der Produkteigenschaft aus einem natürlichen Text extrahiert werden können. Zwar ist das Programm von OntoUSP nicht vorhanden, aber man kann von den Ergebnissen von USP ausgehen, weil OntoUSP eine Erweiterung von USP ist, und bei Experimenten mit Onto USP werden die Ausgaben von USP benutzt, d.h. die Ausgaben von OntoUSP enthalten die gleichen oder ähnliche Informationen wie USP.

1.2 Problem definieren

Die Eingabe ist ein natürliches Dokument. Stanford Parser wandelt das natürliche Dokument in morphologische Wörter und die Dependenzen zwischen den Wörtern um. Die Ergebnisse von Stanford Parser sind die Eingaben für USP. Durch eine Analyse der Ausgaben-Datei des USP Programms wird eine Hierarchie für die Wörter erstellt. Danach werden diese Wörter gefiltert, und die Ausgaben von Produkten, Produkteigenschaften und die Werte der Produkteigenschaften basierend auf folgenden Annahmen generiert :

- Die Wurzel ist entweder ein Produkt oder ein Markenzeichen
- Die Blätter sind die Werte der Produkteigenschaften
- Die Eltern Knoten der Blätter sind Produkteigenschaften

2 Stand der Technik und Grundlagen

2.1 Stand der Technik

in der Vergangenheit wurden viele Ansätze von Extraktion von Produkt und Produkteigenschaftswerten entwickelt. Es gibt auch viele Ansätze für Informationsextraktion basierend auf Ontologie. Aber alle Ansätze erfordern entweder manuelle Unterstützung oder beschränkt sich auf eine bestimmte Wissensdomäne oder dem Format des Textes.

In [8] ist die Extraktion für explizite Produkt Attribute basierend auf der Extraktion für „opinion word“. In [8] sind die manuell markierten Trainingsdaten sind erforderlich. Mit den markierten Daten werden die Pattern erzeugt, und diese Pattern werden wieder benutzt, um nach den „Attribute-Value“ die Entitäten zu extrahieren. Die Bindung von Produkt Attribute und Produkt Attribute Wert, und diese Bindung wird durch einen Dependenzparser (Minipar [9]) realisiert.

In [10] sucht man nach den sehr oft vorkommenden Normen oder Norminalphrasen, die gefundene Normen bzw. Norminalphrasen werden als die Features der Produkte ausgewählt. Mit der Hilfe von PMI(pointwise mutual information) und „Naive Bayes Classifier“ werden die Regeln für die Informationsextraktion generiert.

In [11] werden die Sätze bzw. die Rekorder in Parser Bäume mit leichten semantischen Annotationen umgewandelt, weshalb sich dieser auf eine bestimmte Domäne beschränkt.

In [12] wird die Shallow semantische Analyse benutzt. Die Ontologie der Domäne wird vordefiniert. Die Generierung der Ontologie ist die Abbildung zwischen den Wörtern und der vordefinierten Ontologie, d.h. die Ontologie wird nicht von Wörtern generiert.

In [13] werden die Sätze in einem Text in semantische Term Graph umgewandelt, mit der Hilfe des „Page Ranking Algorithmus“ werden die Term Kandidaten auf entsprechende vordefinierte „Layer“ abgebildet. Gleich wie in [13], wird die Ontologie nicht von Wörtern generiert.

USP liefert vollständig semantische Analysen und ein Markov Logik Netzwerk für Wörter. Damit werden die o.g. Schwächen überwunden.

2.2 Grundlagen

Die syntaktischen Analysen zusammen mit den semantischen Analysen liefern die semantischen Repräsentationen für einen natürlichen Text. Aus einem natürlicher Text werden durch die syntaktische Analyse(eng : syntax parsing) die syntaktische Bäume(auch Parser Bäume) erzeugt. Diese syntaktische Analyse erledigt in USP System durch Stanford Parser. Die Ausgaben von Stanford Parser sind die Eingaben für USP. USP führt die semantische Analyse durch und fügt für jedes Wort eine semantische Rolle hinzu und clustert die Wörter. Die

Ausgaben von USP sind „*.parse“ und „*.mln“ Dateien, wobei die „*.parse“ eine Baum Struktur liefert, und „mln“ ein Netz der Wörter liefert. Durch „*.parse“ und „*.mln“ Datein werden die Ausgaben generiert.

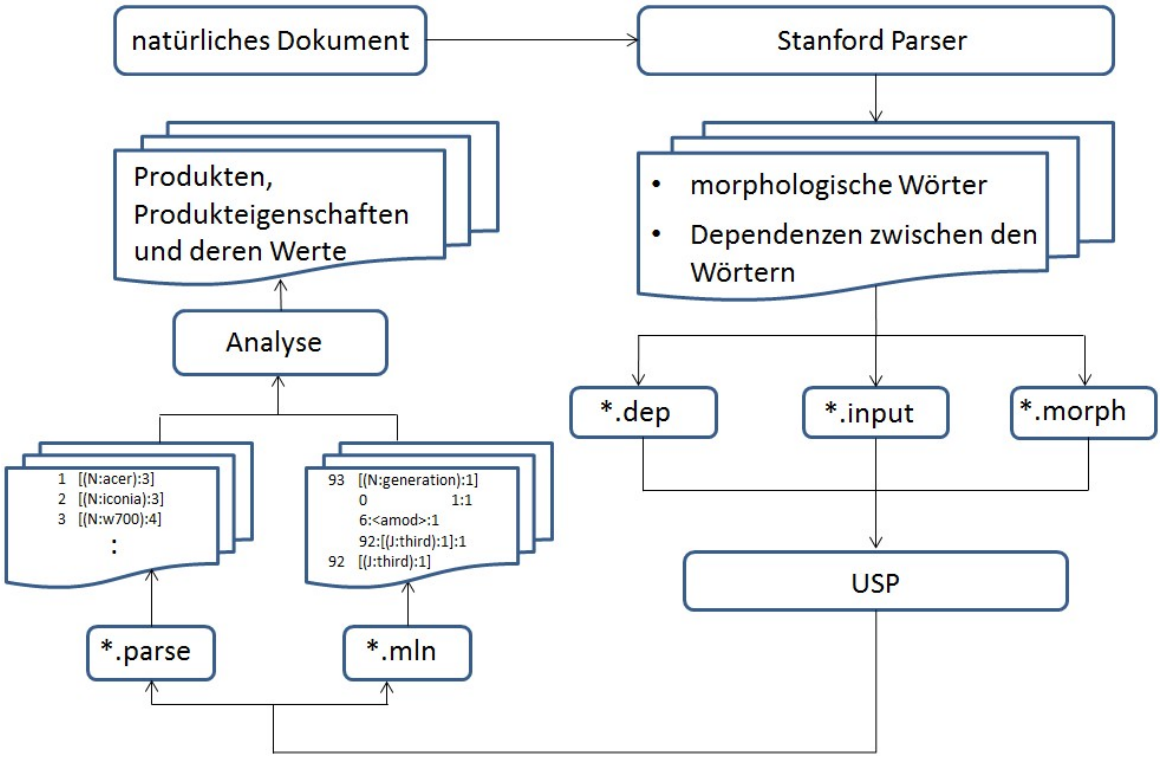


Abbildung 3 : Der Durchlauf der Informationsextraktion

Im Folgend werden die Grundlagen für USP System sowie die Komponente für USP System, Grundlagen für Logik, Grundlagen für Markov Logik Netzwerk und Grundlagen für Liguistik besprochen.

2.2.1 Stanford Parser

„Der Stanford Parser ist ein probabilistisches Parser Programm. Die natürliche Sprache ist die Eingabe für das Programm und durch den probabilistischen Parser kann die grammatische Struktur der Sätze bestimmt werden. Die Kenntnisse der Wahrscheinlichkeiten von probabilistischen Parsern werden aus manuellen analysierten Sätzen erworben, und mithilfe der probabilistischen Parser können die wahrscheinlichste Analyse von neuen Sätzen produziert werden. Dieser probabilistischen Parser garantiert nicht, dass die Ergebnisse 100% richtig sind.“ [14] „*.dep“, „*.input“ und „*.morph“ sind die Eingabe Dateien für USP und die drei Dateien werden von Stanford Parser generiert. In der Tabelle 1 sind ein Beispielsatz und der Inhalt der von Stanford Parser erzeugten Dateien dargestellt. In Abbildung 4 ist die graphische Darstellung der syntaktischen Analyse von Stanford Parser für den Beispielsatz. In Abbildung 5 ist die graphische Darstellung der manuell syntaktischen Analyse für den Beispielsatz gegeben. Stanford Parser hat die syntaktische Struktur falsch analysiert, die Fehler wird mit rot Oval

markiert. Die syntaktische Struktur der Nominalphrase aus Stanford Parser ist NP→NP PP_with NP, die richtige ist NP→NP CONJ_and NP. Solche Fehler führen zu einer falschen Ontologie.

1	Apple announces 13 MacBook Pro with Retina display and new iMac.
2	nsubj(announce-2, Apple-1), num(pro-5, 13-3), nn(pro-5, MacBook-4), dobj(announce-2, pro-5), nn(display-8, Retina-7), prep_with(announce-2, display-8), amod(imac-11, new-10), conj_and(display-8, imac-11)
3	Apple_NNP, announces_VBZ, 13_CD, MacBook_NNP, Pro_NNS, with_IN, Retina_NNP, display_NN, and_CC, new_JJ, iMac_NN, ._.
4	Apple, announce, 13, MacBook, pro, with, Retina, display, and, new, imac, .

Tabelle 1 : Beispiel für Stanford Parser

In der ersten Zeile von Tabelle 1 ist der Satz. In der 2. Zeile sind die Darstellungen von Datei „*.dep“. Die Beziehungen der Wörter (wie nsubj, dobj, nn) und die Position der Wörter („Apple-1“ bedeutet „Apple“ ist das erste Wort, sowie „announce-2“ ist das zweite, usw.) werden angegeben. In der 3. Zeile sind die Darstellungen von Datei „*.input“. Hier werden die Wörter und der entsprechende syntaktische Worttypen angegeben. In Zeile 4 sind die Darstellungen von Datei „*.morph“. Die morphologischen Wörter werden angegeben, z.B. : „announce“ ist das morphologische Wort „announces“ ist das morphologische Wort für „announces“.

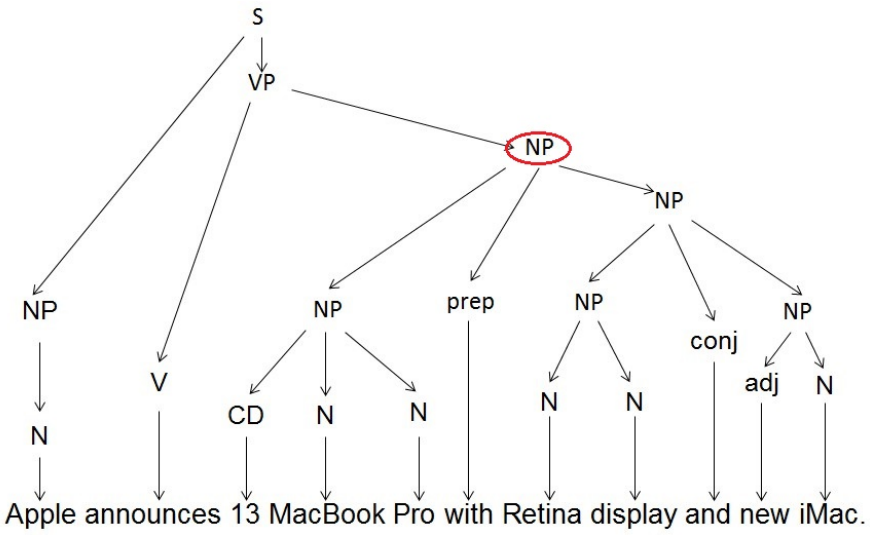


Abbildung 4 : Der Parser Baum, der von Stanford Parser erzeugt wird.

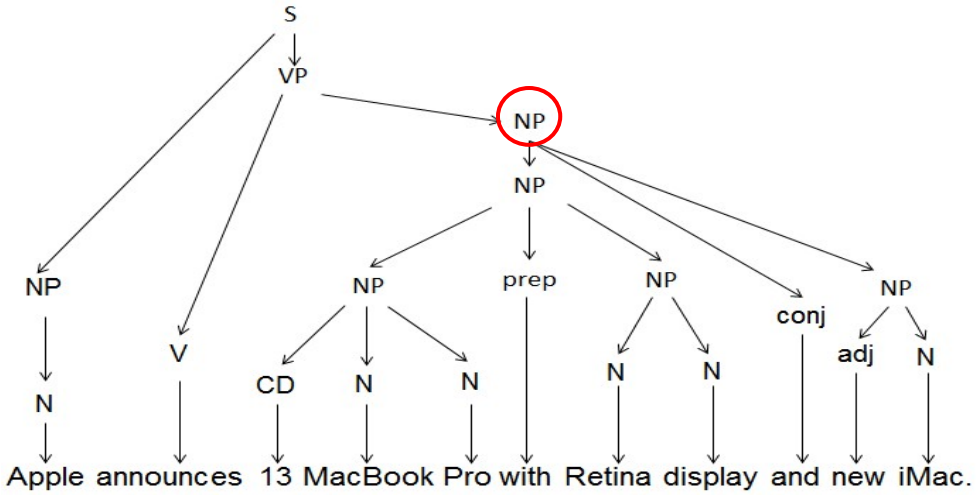


Abbildung 5 : Der Parser Baum, der manuell erzeugt wird.

2.2.2 Semantik Analyse

Für die Verarbeitung natürlicher Sprache gibt es folgende Schwierigkeiten : die heterogenen Wissensdomänen, die Auflösung der Ambiguität der Sprache, Modellierung der Sprache und die Auflösung der syntaktischen Variationen der Sprache. Im Durchlauf von „Shallow Semantic Parsing“ erfolgt die Erkennung von semantischer Repräsentation z.B. „Wer“, „Was“, „Wann“, „Wo“, „Warum“, „Wie“, usw. und nur die Elemente, die aufeinander folgend und zusammenhängend sind, werden ermittelt. „Shallow Semantic Parsing“ kann die s.g. Schwierigkeiten nicht auflösen und es fehlt die Fähigkeit der Schlussfolgerung.

Abbildung 6 : Ein Beispiel für Shallow Semantic Parsing

Im Gegensatz zu „Shallow Semantic Parsing“ liefert die vollständige semantische Analyse eine Repräsentation eines Satzes in Prädikatenlogik erster Stufe oder andere formale Sprache und unterstützt eine automatische Schlussfolgerung. Die natürliche Sprache wird im Leseprozess vollständig syntaktisch analysiert, damit die logischen Formen erzeugt werden können. Durch semantische Analyse wird die von syntaktischer Analyse erzeugte logische Form, nämlich die logische Repräsentation der natürlichen Sprache, auf der vollständigen semantischen Repräsentation, nämlich die Bedeutungsrepräsentation, abgebildet. Die Bedeutungsrepräsentationen der Sprache werden in dieser Ausarbeitung durch Prädikatenlogik erster Stufe dargestellt. Einige Definitionen müssen hier angeben werden :

- Ein Term ist ein Objekt in einer Domäne. Ein Term kann eine Konstante, eine Variable oder eine Funktion, die auf den Variablen angewendet, sein.

- Eine Formel bzw. eine atomare Formel ist ein Prädikatsymbol, das auf n-Tupel von Termen angewendet wird. Eine Formel kann aus mehreren atomaren Formeln verknüpft mit logischen Symbolen und Quantoren rekursiv konstruiert sein.
- Unter einer lexikalischen Einheit kann in dieser Ausarbeitung ein Wort verstanden werden. Ein lexikalischer Eintrag definiert die logische Formel für eine lexikalische Einheit mit POS Tagging. λ-gebundene Variablen markieren die fehlenden Argumenten in den logischen Formen.

„Der semantische Parser eines Satzes wird hergeleitet, indem man mit logischen Formen in den lexikalischen Einträgen anfängt und die Bedeutung größerer Fragmente rekursiv aus deren Bestandteilen zusammensetzt.“ [7] In der Abbildung 7 wird gezeigt, dass „everybody“, „two language“ und „speaks“ zuerst analysiert werden, danach werden die kleinen Bedeutungsrepräsentationen in einer großen Bedeutungsrepräsentation zusammengestellt.

$$\text{Verb}[\lambda y\, \lambda x.\text{love}(x,y)] \rightarrow \text{loves}$$
$$\text{NP}[\text{Tom}] \rightarrow \text{Tom}$$
$$\text{NP}[\text{Mary}] \rightarrow \text{Mary}$$
$$\text{VP}[\text{rel}(\text{obj})] \rightarrow \text{Verb}[\text{rel}]\ \text{NP}[\text{obj}]$$
$$\text{S}[\text{rel}(\text{obj})] \rightarrow \text{NP}[\text{obj}]\ \text{VP}[\text{rel}]$$

Abbildung 7 : Ein Beispiel für semantische Analyse

Die ersten drei Zeilen in der Abbildung 7 sind lexikalische Einträge bzw. Wörter. Die syntaktische Kategorie bzw. POS Tagging von „love“ ist „Verb“. Wenn zwei Atome die Funktion „loves(x,y)“ erfüllen, dann ist diese Funktion true. Die letzten zwei Zeilen haben gezeigt, dass die lexikalischen Einträge in einem größeren Fragment der Bedeutung zusammengestellt werden.

2.2.3 Prädikatenlogik erster Stufe (Abk. PL1)

„Die Prädikatenlogik erster Stufe beschäftigt sich mit Objekten und Aussagen über deren Eigenschaften.“ [15] Die Prädikatenlogik erster Stufe ist „ausdruckstärker“ als Aussagenlogik, und Quantoren, Funktions- und Prädikatsymbole kommen hinzu. Die Prädikatlogik erster Stufe ermöglicht ontologische Bindung zwischen den Objekten, das ist der wichtigste Unterschied von allen zwischen Aussagenlogik und Prädikatlogik erster Stufe.

Einige Definitionen (aus [16]) :

- Eine Variable hat die Form x_i
- Ein Prädikatsymbol hat die Form Pi und ein Funktionssymbol hat die Form f_i
- Jede Variable ist ein Term, jede Konstante ein Term, sowie f (t_1,…t_n) auch ein Term, falls f eine Funktion und t_i die Terme sind.
- P(t_1,…t_k) ist eine Formel bzw. eine atomare Formel, fall P ein Prädikatsymbol ist und t_i Terme sind.

- Für jede Formel ¬F, ∃F, ∀F, F∨G, F∧G sind auch die Formeln, wobei ∃ und ∀ sind die Quantoren.
- Alle vorkommenden Variablen sind entweder frei oder gebunden. Wenn x in der Form ∃xF oder ∀xF vorkommt, dann heißt die Variable x in Formel F gebunden, andernfalls heißt frei.

Eine Wissensdomäne ist eine Sammlung von Informationen über die Bedeutungen der Daten und über die logischen Regeln. Eine PL1 Wissensdomäne ist eine Menge von Sätzen und Formeln in PL1. Die Formeln bestehen aus Konstante, Variablen, Funktionen und Prädikaten. Die neue Regeln bzw. die Randbedingungen können in die vorhandene Wissensdomäne hinzugefügt werden, und die Wissensdomäne mittels der Regeln bzw. die Randbedingungen aus dem vorhandenen Wissen Schlüsse inferieren.

Einige Definitionen (aus [17]) :

- Jede Konstante ist ein Objekt in einer Wissensdomäne und kann typisiert sein, z.B. Konstant HA repräsentiert Hersteller Apple.
- Die Variablen können typisiert sein und repräsentieren die Objekte gleichen Typs in einer Wissensbasis, z.B. Variable x_i = Tom ist der Name der Menschen in der Wissensbasis „MenschenName (Tom, Jerry, Mary)“. Durch die Substitution der Variablen durch eine Konstante aus der Konstante Menge werden die verschiedenen Objekte entstehen, z.B. MenschenName (Tom) und MenschenName (Jerry).
- Die Beziehung zwischen den Objekten sind die Funktionen wie Mutter_von, guter_Freund_von.
- Ein Prädikatsymbol repräsentiert die Beziehung zwischen den Objekten und die Eigenschaften der Objekte, z.B. Feind und Rauchen.
- Ein Term kann eine Konstante, eine Variable oder eine Funktion sein, die auf Unterterme angewendet werden kann.
- Eine atomare Formel ist ein Prädikatsymbol, das auf Unterterme angewendet wird.
- Eine Formel kann rekursiv aus atomaren Formeln, die mit Quantoren(∃,∀) und logischen Symbolen(∧,∨,⇔,⇒) der Prädikatenlogik verknüpft sind, konstruiert werden.
- Ein Grundterm ist ein Term, der keine Variable enthält.
- Ein Grundatom oder ein Grundprädikat ist eine atomare Formel, deren Argumente alle Grundterme sind.

Die von Stanford Parser erzeugten Dependenzen können in die PL1 Repräsentationen umgewandelt werden. Der natürliche Text kann als die Kombination von den PL1 Repräsentationen gesehen werden. Somit ist die Wissensdomäne eines Textes die Sammlung von Dependenzen in PL1 Repräsentationen mit den Objekten und diese PL1 Repräsentationen sind die Randbedingungen im Markov Logik Netzwerk. Die Randbedingungen im Markov Logik Netzwerk sind die in disjunktiver Normalform geschriebenen Klausel-Formen, damit die Randbedingungen aufgeweicht werden. Eine Welt (possible world) ist wahr, wenn alle vorkommenden Grundatome wahr sind.

English	First-Order Logic	Clausal Form
Friends of friends are friends.	$\forall x \forall y \forall z\ \mathtt{Fr}(x,y) \wedge \mathtt{Fr}(y,z) \Rightarrow \mathtt{Fr}(x,z)$	$\neg\mathtt{Fr}(x,y) \vee \neg\mathtt{Fr}(y,z) \vee \mathtt{Fr}(x,z)$
Friendless people smoke.	$\forall x\ (\neg(\exists y\ \mathtt{Fr}(x,y)) \Rightarrow \mathtt{Sm}(x))$	$\mathtt{Fr}(x,\mathtt{g}(x)) \vee \mathtt{Sm}(x)$
Smoking causes cancer.	$\forall x\ \mathtt{Sm}(x) \Rightarrow \mathtt{Ca}(x)$	$\neg\mathtt{Sm}(x) \vee \mathtt{Ca}(x)$
If two people are friends, either both smoke or neither does.	$\forall x \forall y\ \mathtt{Fr}(x,y) \Rightarrow (\mathtt{Sm}(x) \Leftrightarrow \mathtt{Sm}(y))$	$\neg\mathtt{Fr}(x,y) \vee \mathtt{Sm}(x) \vee \neg\mathtt{Sm}(y)$, $\neg\mathtt{Fr}(x,y) \vee \neg\mathtt{Sm}(x) \vee \mathtt{Sm}(y)$

Abbildung 8 : (aus [17]) Beispiele für Wissensdomäne von Prädikatenlogik erster Stufe. Fr() ist Abkürzung für Friends(), Sm() für Smokes(), and Ca() für Cancer().

2.2.4 Quasi-logische Form, POS Tagging und Lambda(-Kalkül) Notation

2.2.4.1 Quasi-logische Form

Quasi-logische Form basiert auf der Prädikatlogik (in dieser Ausarbeitung wird mit Prädikatenlogik erster Stufe beschäftigt. Details siehe [18]) und ist eine Darstellung der Bedeutung des Dokuments. Jede QLF hat eine entsprechende Formel in Prädikatenlogik bzw. Prädikatenlogik erster Stufe.

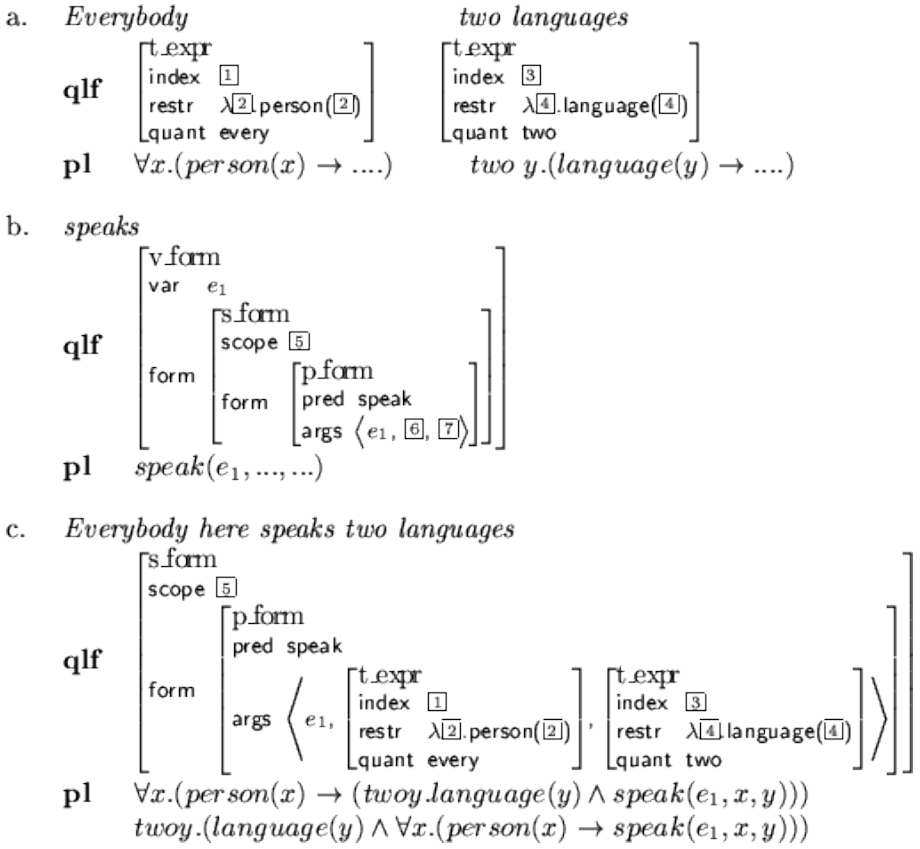


Abbildung 9 (aus [19]): Die Umwandelung des Satzes „Everybody speaks two languages“ in QLF und entsprechender Prädikatenlogik

2.2.4.2 Lambda(-Kalkül) Notation

„Durch Lambda(λ)-Ausdrücke werden formale Parameter eingeführt, die durch Terme ersetzt werden können.“ [20] Beide λx.Love(x, y) und λx. λy.Love(x, y) sind die Lambda-Ausdrücke. Die λ-gebundene Variable kann durch ein Argument aus einem Definitionsbereich, z.B. aus konstanten Menge in MLN, substituiert werden. Die nicht-λ-gebundene Variable heißt frei. Ein Beispiel für Substitution einer λ-gebundenen Variable :

$$\lambda x.\text{Love}(x, y)(\text{Tom}) \Rightarrow \text{Love}(\text{Tom}, y)$$

wobei y eine nicht-λ-gebundenen Variable bzw. eine freie Variable ist. Die Substitution von λ-gebundenen Variable ist die λ-Reduktion. „Ein Prädikat mit mehreren Argumenten kann durch die λ-Reduktion auf eine Folge von jeweils einstelligen Prädikaten abbilden.“ [20]

2.2.4.3 POS Tagging

Ein Token ist in dieser Ausarbeitung ein einzelnes Wort. Ein „Tag“ ist eine Markierung bzw. eine Etikett von Token. Tagging ist ein Verfahren, durch das ein Tag einem Token zugewiesen wird. POS(PART-OF-SPEECH) Tagging ist die Zuordnung der Wortart zu einem Token, z.B. Verb, Nomen usw. Mithilfe von POS Tagging werden die Informationen der Sprache kategorisiert. Die von Stanford Parser erzeugten Dependenzen enthalten die POS Tags schon.

Abbildung 10 : Ein Satz markiert mit POS Tag.

2.2.5 Markov Logik Netzwerk (Abk. MLN)

2.2.5.1 Markov Netzwerk (Abk. MN)

Das Folgende baut auf [17] auf. „Markov Netzwerk (oder Markov Random Field) ist ein statistisches Modell für multivariate Verteilung einer Menge von Variablen $X = (X_1, X_2, \dots, X_n) \in \aleph$ und beschreibt die ungerichteten Graphen, die bedingte Unabhängigkeitsaussagen zwischen Variablen ausdrucken.“ [17] & [21] In einem ungerichteten Graph repräsentiert jeder Knote eine Variable, jede Clique im Graph hat eine potenzielle Funktion, die einen Zustand der Clique repräsentiert. Die multivariate Verteilung von MN :

$$P(X = x) = \frac{1}{z} \Pi \Phi_k\left(x_{\{k\}}\right) \quad (1)$$

wobei x{k} ist der Zustand von k-ste Clique ist, d.h. x{k} repräsentieren alle Werte von den Variablen in der k-ste Clique. Z ist die Normalisierung. Die Formel (1) kann als log-lineares Modell dargestellt werden.

$$P(X = x) = \frac{1}{z}\exp\left\{\sum_j \omega_j f_j(x)\right\} \quad (2)$$

wobei ω_j Gewicht von einer Feature Funktion $f_j(x)$ ist. In dieser Ausarbeitung $f_j(x) \in \{0,1\}$. Feature Funktion beschreibt einen Zustand der Clique und das Gewicht der Feature Funktion ist $\log_{\varphi_k}(x_{\{k\}})$.

2.2.5.2 Markov Logik Netzwerk (Abk. MLN)

Das Folgende baut auf [17] auf. Eine Wissensdomäne besteht aus einer Folge von logischen Regeln der Prädikatenlogik erster Stufe. Eine logische Regel in einer Wissensdomäne ist eine Randbedingung in einer Welt. Wenn eine Randbedingung einer Welt verletzt ist, hat die entsprechende Welt die Wahrscheinlichkeit 0. Sehr oft ist ein solcher Fall nicht erwünscht. Mit Markov Logik wird diese Randbedingung dadurch aufgeweicht, dass jeder Randbedingung ein Gewicht zugewiesen wird. Damit ist es möglich, dass eine Welt, in der eine Randbedingung verletzt ist, eine geringere Wahrscheinlichkeit besitzt, aber nicht unmöglich ist. Das Gewicht beschreibt die Bindungswirkung der entsprechenden Randbedingung, wenn die Welt wahr ist. Eine Welt, in der die Randbedingung erfüllt ist, besitzt größere Wahrscheinlichkeit als eine alternative Welt, in der die Randbedingung verletzt wird. „Ein Markov Logik Netzwerk ist eine probabilistisch logische Repräsentation, welche Prädikatenlogik erster Stufe und Markov-Netze miteinander verknüpft.“ [22]

Einige Definition (aus [17]) :

- Ein Markov Logik Netzwerk L ist eine Menge von Paaren (F_i, ω_i), wobei F_i eine logische Formel der Prädikatenlogik der erster Stufe ist und ω_i ein Gewicht. Hier werden nur die existenzquantifizierte Variablen betrachtet und alle allquantifizierten Variablen sind die freie Variablen.
- (F_i, ω_i) zusammen mit einer endlichen Menge von logischen Konstanten $C = \{c_1, c_2, \dots, c_{|c|}\}$ definieren ein Markov Logik Netz $M_{L,C}$.
- Jedes Grundatom in L entspricht einem binären Wert Knoten in $M_{L,C}$. Die Grundatome Menge $X= \{X_1,\dots,X_n\}$ wird dadurch erhalten, dass die Variablen der prädikatenlogischen Formel in L durch die in L gegebenen Konstanten substituiert werden. Der Wert eines Knotens ist genau dann 1, wenn das Grundatom wahr ist, ansonsten ist der Wert 0. Der Knoten kann mit anderen Knoten durch die Kanten verbunden werden, wenn die beiden Knoten bzw. die Grundatome in einer Belegung der Grundformel gemeinsam vorkommen.
- Für jede Belegung einer Grundformel F_i in L besitzt ein Feature f_i, der Wert von f_i genau dann 1 ist, wenn die Belegung der Grundformel wahr ist und sonst 0. Die Summe der Gewichte für Feature f_i ist das Gewicht w_i in L.

English	First-Order Logic	Clausal Form	Weight
Friends of friends are friends.	$\forall x \forall y \forall z\ \mathtt{Fr}(x,y) \wedge \mathtt{Fr}(y,z) \Rightarrow \mathtt{Fr}(x,z)$	$\neg\mathtt{Fr}(x,y) \vee \neg\mathtt{Fr}(y,z) \vee \mathtt{Fr}(x,z)$	0.7
Friendless people smoke.	$\forall x\ (\neg(\exists y\ \mathtt{Fr}(x,y)) \Rightarrow \mathtt{Sm}(x))$	$\mathtt{Fr}(x,g(x)) \vee \mathtt{Sm}(x)$	2.3
Smoking causes cancer.	$\forall x\ \mathtt{Sm}(x) \Rightarrow \mathtt{Ca}(x)$	$\neg\mathtt{Sm}(x) \vee \mathtt{Ca}(x)$	1.5
If two people are friends, either	$\forall x \forall y\ \mathtt{Fr}(x,y) \Rightarrow (\mathtt{Sm}(x) \Leftrightarrow \mathtt{Sm}(y))$	$\neg\mathtt{Fr}(x,y) \vee \mathtt{Sm}(x) \vee \neg\mathtt{Sm}(y)$,	1.1
both smoke or neither does.		$\neg\mathtt{Fr}(x,y) \vee \neg\mathtt{Sm}(x) \vee \mathtt{Sm}(y)$	1.1

Tabelle 2: (aus [17]) Beispiele für Wissensdomäne von MLN.

In der Tabelle 2 ist Fr() Abkürzung für Friends(), Sm() für Smokes(), and Ca() für Cancer(). Im Vergleich zu Tabelle 2 werden hier die Gewichte hinzugefügt. Allen Regeln in dem Spalt „First-Order Logic“ sind die Grundformeln und die Belegungen der Grundformeln in dem Spalt „Clausal Form“. Wenn eine Grundformel mehre Belegungen besitzt, dann wird das Gewicht der Grundformel gleichmäßig auf Belegungen aufgeteilt, z.B. Das Gewicht für ∀x ∀y Fr(x; y) ⇒ (Sm(x) , Sm(y)) ist 2.2 und das Gewichte für jede Belegung ist 1.1.

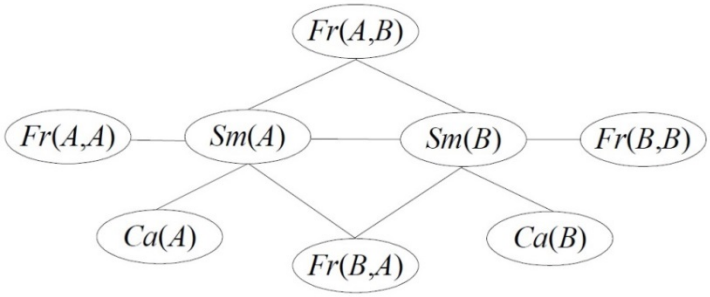


Abbildung 11 : (aus [17]) : Der MLN Graph, der Tabelle 2entspricht. A und B sind die Konstant.

First-Order logic	Variable assignment	Clique	Weight
F_1:$\forall x, Sm(x) \Rightarrow Ca(x)$	x=A	$\{Sm(A), Ca(A)\}$	1.5
	x=B	$\{Sm(B), Ca(B)\}$	1.5
F_2:$\forall x \forall y, Fr(x,y) \Rightarrow (Sm(x) \Leftrightarrow Sm(y))$	x=A, y=A	$\{Fr(A,A), Sm(A)\}$	1.1
	x=A, y=B	$\{Fr(A,B), Sm(A), Sm(B)\}$	1.1
	x=B, y=A	$\{Fr(B,A), Sm(A), Sm(B)\}$	1.1
	x=B, y=B	$\{Fr(B,B), Sm(B)\}$	1.1

Abbildung 12 (aus [23]) : Cliques und Gewichte von MLN in Abbildung 11. Es gibt 6 Cliques.

Aus der Definition von MLN wird jeder Knoten in MLN $M_{L,C}$ durch Einsetzen für Variablen der logischen Formeln in MLN die Grundatome erzeugt. Die Kante zwischen den Knoten entspricht die Beziehung zwischen den Knoten. Deshalb kann MLN als Model von Markov Netzwerk gesehen werden und die Wahrscheinlichkeitsverteilung ist :

$$P(X = x) = \frac{1}{z}\exp\left(\sum_i \omega_i n_i(\chi)\right) = \frac{1}{z}\prod_i \Phi_i\left(x_{\{i\}}\right)^{n_i(x)} \qquad (5)$$

wobei n_i die Anzahl der Grundformeln ist, die den Wert 1 haben. Formel (5) hat gezeigt, dass eine Welt nicht unmöglich ist, wenn diese Welt eine Randbedingung oder mehrere Randbedingungen verletzt, sondern besitzt geringere Wahrscheinlichkeit. Gewicht ω_i zeigt, wie „stark“ die Randbedingung in der Welt ist. Für ein bestimmtes MLN können unterschiedliche Markov Netzwerk erzeugt werden, wenn die Konstant Menge unterschiedlich sind, aber diese Markov Netzwerk haben auch manche Gemeinsamkeiten wie die gleiche Anzahl der Clique. Die Gewichte werden entweder aus dem Lernen von Trainingsdaten erhalten oder von Menschen manuell gegeben.

2.2.6 Clusteranalyse

USP System startet mit Clustering von Wörtern, die gleichen Typen haben, baut rekursiv größere Clusters auf. Hier wird das Cluster erklärt, das Cluster ist eine Gruppe von Objekten, die ähnliche Eigenschaften besitzen. Die Objekte werden entweder in verschiedene Klassen aufgeteilt oder besitzen keine Struktur. Das Clustering (auch Clusteranalyse) dient dazu, dass die Objekte ins Cluster untergeteilt werden, damit die in einem Cluster zugeordneten Objekte eine möglichst hohe Ähnlichkeit besitzen. Die Clusteranalyse ermöglicht eine Struktur für die Objekte aufzubauen. Bei der Clusteranalyse ist das Ziel, die Unterschiede zwischen den einzelnen Gruppen möglichst maximiert und die Unterschiede innerhalb der einzelnen Gruppen möglichst minimiert werden zu können.

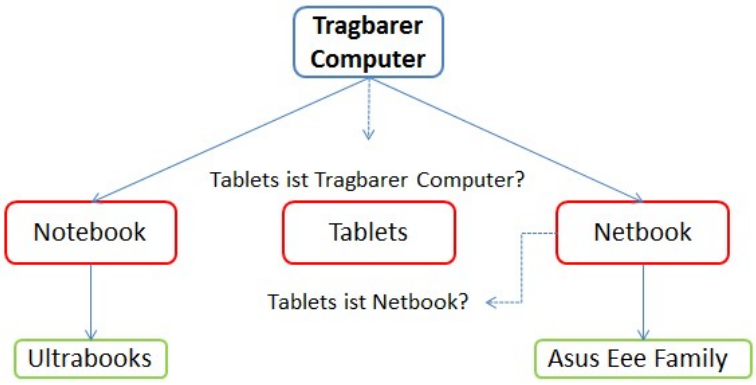


Abbildung 13: Ein Beispiel für Clusteranalyse. „Tablet PC“ kann in der Gruppe „Tragbarer Computer“ oder in „Netbook“ Gruppe sein. In Welcher Gruppe die Objekte zugeordnet werden, hängt stark vom verwendeten Algorithmus, Parametern und verwendeten Objekt-Attributen ab.

2.2.7 Unsupervised Semantic Parsing und Ontology Unsupervised Semantic Parsing

Früher wurde der semantische Parser manuell erstellt, zwar einige Ansätze für maschinelles Lernen wurden danach entwickelt, aber die manuelle Unterstützung war immer noch erforderlich, und manche Ansätze beschränkten sich auf einer geschlossenen Wissensdomäne. USP ist der erste nicht-überwachte maschinelles Lernen Ansatz für Semantik Parser. Ob ein

Ansatz für maschinelles Lernen überwacht ist, ist abhängig davon, ob Eingabe- und Ausgabedatei manuell markiert sind.

- überwachtes Lernen (engl. supervised learning) : Der Algorithmus lernt eine Funktion aus gegebenen Paaren von Ein- und Ausgaben. Dabei stellt während des Lernens ein „Lehrer“ den korrekten Funktionswert zu einer Eingabe bereit.
- nicht-überwachtes Lernen (engl. unsupervised learning) : Der Algorithmus erzeugt für eine gegebene Menge von Eingaben ein Modell, das die Eingaben beschreibt und Vorhersagen ermöglicht. Dabei gibt es Clustering-Verfahren, die die Daten in mehrere Kategorien einteilen, die sich durch charakteristische Muster voneinander unterscheiden.

2.2.7.1 Unsupervised Semantic Parsing

USP beruht auf drei zentralen Ideen : (aus [7])

- Ziel Prädikat und Objekt Konstanten können als Cluster von syntaktischen Variationen derselben Bedeutung angesehen werden, und aus Daten erlernt werden. Zum Beispiel stellt „ACQUIRE“ den Erwerb Beziehung, und kann als Cluster von verschiedenen Formen zum Ausdruck dieser Beziehung, wie „acquired“, „bought“, „purchased“ angesehen werden; Microsoft repräsentiert das Unternehmen Microsoft und kann als das Cluster von „Microsoft“, usw. angesehen werden.
- Die gleiche Formen können clustert werden. Die Formen, die aus den gleichen Formen bestehen, können clustert werden.

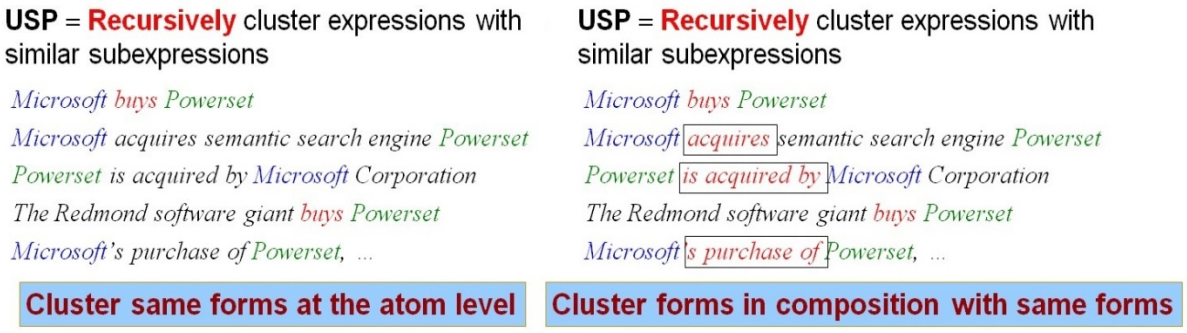


Abbildung 14 (aus [24]) : Illustration für Clustering-Verfahren von USP

- USP startet direkt von syntaktischen Analysen und konzentriert sich nur auf deren Umsetzung zum semantischen Inhalt. Die vorherige entwickelte Parser können in USP eingesetzt werden, deshalb stehen viele Ressourcen zur Verfügung. Die syntaktische Analyse und die semantische Analyse sind in USP getrennt, damit die Komplexität der semantischen Analyse reduziert wird, weil es nicht erforderlich ist, bei Zusammensetzung der Bedeutungen ein domänenspezifisches Verfahren zur Erzeugung von Kandidaten Lexikon zu brauchen.

Die Eingaben für USP System sind die Dependenzbäume, die von Stanford Parser generiert werden. Eine natürliche Sprache ist die Eingabe für Stanford Parser, „*.dep“, „*.input“ und „*.morph“ sind die Ausgabe von Stanford Parser, aus diesen drei Dateien werden die Dependenzbäume generiert. Ein Dependenzbaum ist eine Baum Struktur, in der Baum-Struktur

sind die Knoten die Wörter, und die Kanten sind die Beziehungen zwischen den Wörter. Die QLF in dieser Ausarbeitung wird daraus erhalten :

- jeder Knoten in einem Dependenzbaum wird in ein unäres Atom mit dem Prädikat umwandelt, und das Prädikat wird mit POS Tag markiert, z.B. „Microsoft“ wird in „Microsoft(n4)“ umgewandelt.
- jede Kante in einem Dependenzbaum entspricht einem Prädikat mit zwei Atomen, und das Prädikat ist die „Beziehung“ zwischen den Atomen, z.B. nsubj(n3, n2).

wobei n_2 und n_3 die Skolemkonstante sind. Eine Skolemkonstante kann man in dieser Ausarbeitung als eine Konstante aus der konstanten Menge in MLN verstehen. (Mehr über Skolemkonstante sieht Kapitel 9 in [25]).

QLF von einem Satz kann als eine Konjunktion von logischen Formen von entsprechenden Konten und Kanten angesehen werden. Dieser Vorgang ist illustriert in Abbildung 15. Die Bedeutung eines Satzes kann als eine Kombination von den Sub-Formen der QLF gesehen werden, deshalb die lexikalischen Einträge beschränken sich nicht mehr auf den adjazenten Wörtern sondern sind die beliebigen Fragmente in einem Dependenzbaum. Deswegen hat USP System mehre Flexibilität beim Maschinenlernen.

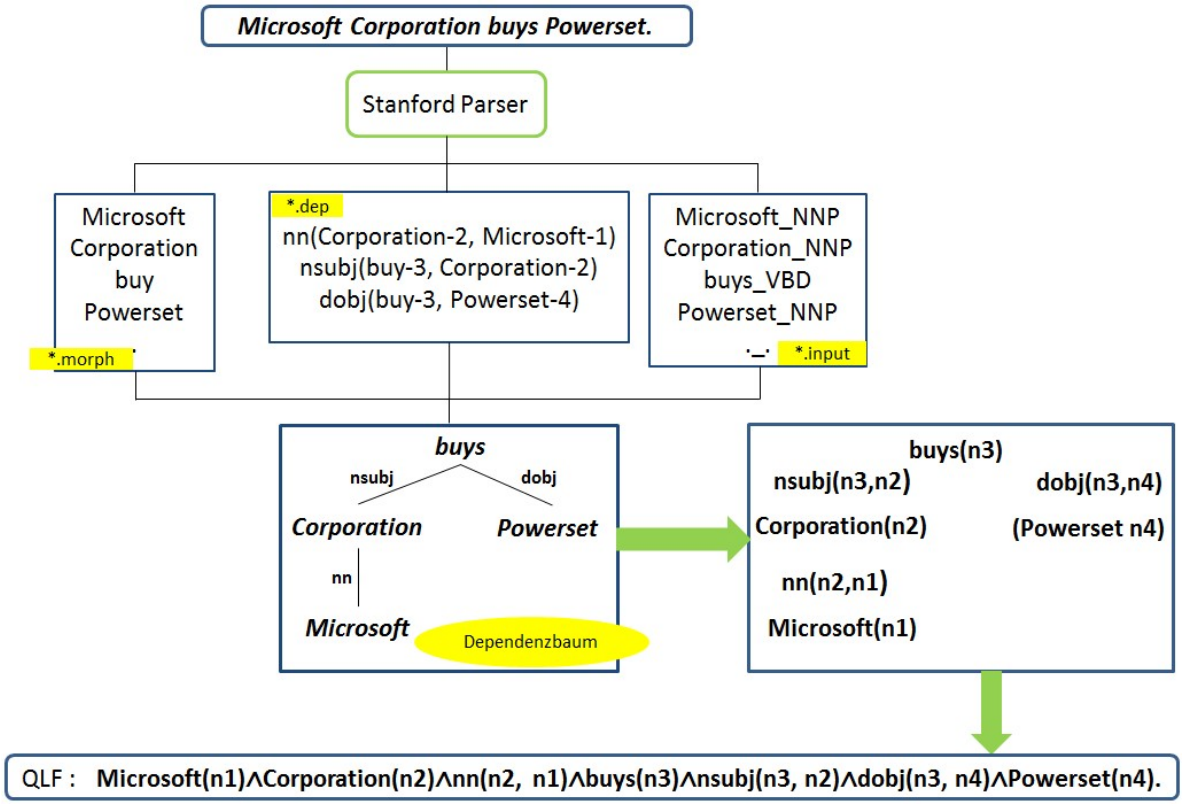


Abbildung 15: Beispiel für die Generierung von QLF.

Die natürliche Sprache wird durch s.g. Stanford Parser drei Dateien als Eingaben für USP System erzeugt. Diese drei Dateien stellen einen Dependenzbaum dar. Der Dependenzbaum

wird in QLF umgewandelt. QLF in dieser Ausarbeitung wird vereinfacht. Deshalb ist die Darstellung hier nicht gleich wie die Darstellung in der Abbildung 9.

In dem USP System wird QLF in den kleinen Teilen partitioniert, und die partitionierte Teile der QLF werden in einer Gruppe bzw. in einem Cluster zugewiesen, wenn sie die gleiche Bedeutung besitzen, z.B. „Microsoft“ und „Corporation“ besitzen die gleiche Bedeutung.

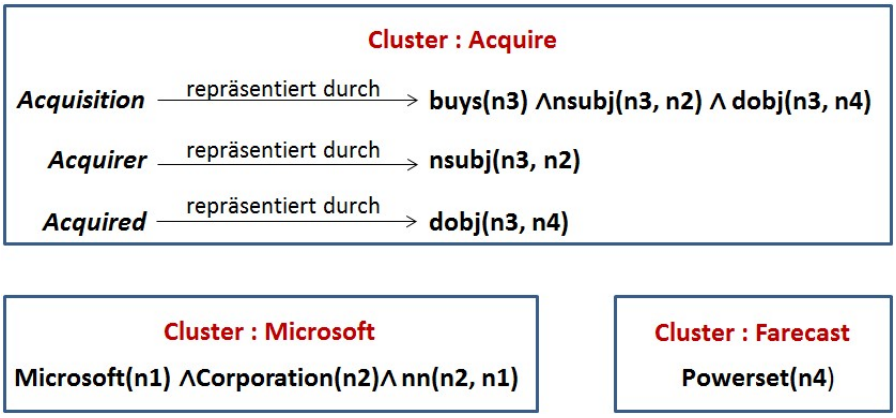


Abbildung 16 : Die Partitionen von QLF. Wenn die Atomen die gleichen Bedeutungseinheit besitzt, werden die Atomen in einem Cluster bzw. in einer Partition hinzufügt.

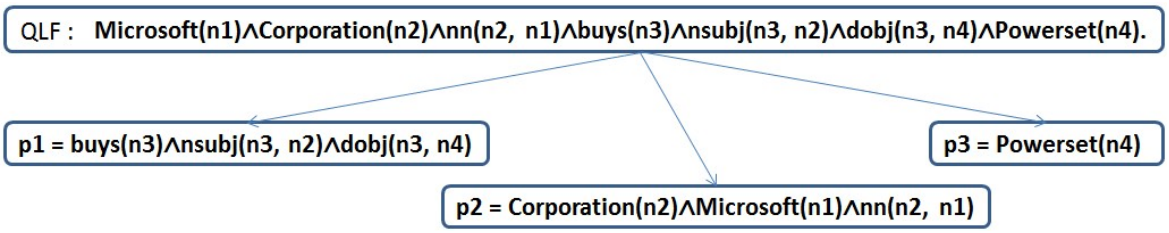


Abbildung 17 : Die sub-Formeln von QLF

Manche Atome sind die Bedeutungseinheiten, und mache Atome sind die Argumente. Z.B. : **buys(n3)** ist ein „ACQUIRE Event“, **Corporation(n2)** ist ein Argument für „ACQUIRER“ Beziehung von **nsubj(n3, n2),** und **Powerset(n4)** ist ein Argument für „ACQUIRED“ Beziehung von **dobj(n3, n4).** In USP System hat jede Sub-Formel von QLF die entsprechende Lambda Form. Bei der zugehöriger Lambda-Form wird jede Konstante n_i, die nicht in einem einstelligen Atom von Formel F vorkommt, durch eine eindeutige Variable x_i ersetzt. Z.B. :

buys(n3)∧nsubj(n3, n2)∧dobj(n1, n4)

$$\lambda x_2 . \lambda x_4 .\textbf{buys(n3) ∧ nsubj(n3, } x_2\textbf{) ∧ dobj(n3, } x_4\textbf{)}$$

wobei n_1, n_2 und n_3 die Skolemkonstante sind.

Die Lambdaform wird mit Hilfe von Davidsonian Semantics weiter zerlegt in Core Form und Argumentform. Die Core Form ist eine Lambdaform, die keine Lambdavariable enthält, und eine Argumentform ist eine Lambdaform, die nur eine Lambdavariable enthält.

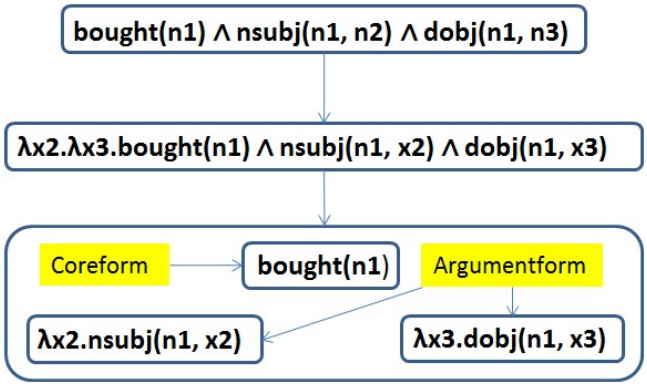


Abbildung 18 : Ein Beispiel für Erzeugung der Lambdaformen

Durch die Clusteranalyse werden die Lambdaformen in den Lambda-Form Cluster aufgeteilt. Ein Lambda-Form Cluster ist ein Cluster, das die semantisch austauschbaren Lambdaformen enthält und die Bedeutung der Sub-Formel von QLF. Lambdaform Cluster kann den Argumenttypen enthalten, damit die Typen der Argumente in den Beziehungen unterschieden werden können. Z.B. : die Argumente „ACQUIRER“ und „ACQUIRED“ in den Relationen **nsubj(n1, n2)** und **dobj(n1, n3)** entsprechen dem Subjekt und Objekt von Verb „buys“. In Stanford Parser kann die Argument „ACQUIRED“ als „nsubjpass“ für ein Subjekt in einen passiven Satz repräsentiert werden. Die syntaktischen Variationen werden in dem Lambda-Form Cluster abgezogen und unterschieden sich durch den Argumenttypen.

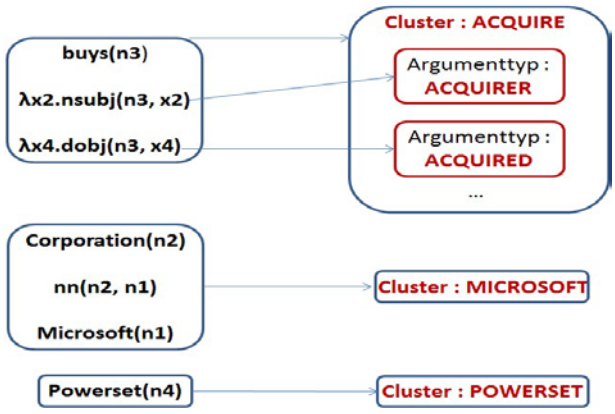


Abbildung 19 : Die Lambdaformen werden auf den Cluster aufgeteilt und den syntaktischen Variationen in Argumenttypen zugeordnet. Links sind die Lambdaformen und rechts sind die Cluster.

Im USP System startet die semantische Analyse mit Clusteranalyse der Lambda-Formen in Token bzw. in Atom Ebene, d.h. die QLFs werden durch die Partition auf den Sub-Formen der QLF abgebildet, jede Sub-Form hat eine entsprechende Lambdaform, die Lambdaform wird weiter in Core Form und Argumentform zerlegt. Um die Argumentform zu unterscheiden, wird jede Argumentform einem Argumenttyp zugewiesen. Eine Regel, bei der eine Lambda-Form auf ein Cluster abgebildet wird und einen Argumenttyp zuweist, ist eine semantische Grammatik. Mit der semantischen Grammatik werden dann die Core Formen auf die Cluster und die

Argumentformen auf den Argumenttypen abgebildet. USP eine Wahrscheinlichkeitsverteilung über den semantischen Parser. Das Problem von maschinellem Lernen in USP ist das Lernen von s.g. semantischer Grammatik. Das Lernen in USP wird realisiert durch die Nutzung von Markov Logik Netzwerk.

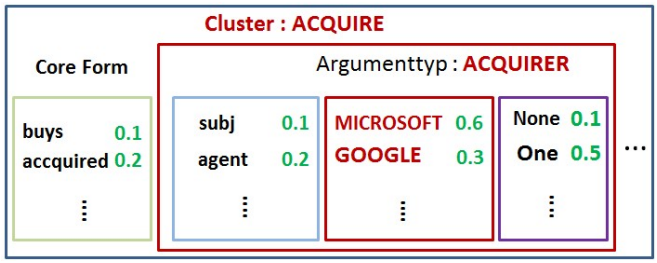


Abbildung 20 : Beispiel für ein Cluster

Ein semantischer Parser L partitioniert ein QLF in die QLF-Teile p_1, p_2, … , p_n, jedes Teil p wird in einen oder einige Lambdaform Cluster c zugewiesen, und p_i wird später in Core Form f und Argumentformen f_1, f_2, … , f_k umgewandelt, jede Argumentform besitzt auch einen Argumenttyp a in c. Um die Verteilung über die Lambdaformen zu modellieren, werden Form(p, f!), ArgForm(p, i, f!) definiert, wobei p eine Partition ist, i der Index eines Arguments und f eine Sub-Formel von QLF. Form(p, f!) ist true genau dann, wenn Partition p eine Core Form f hat, und ArgForm(p, i, f!) ist true genau dann, wenn i-stes Argument in p die Sub-Form hat. „f!“ Notation bedeutet, dass jede Partition oder jedes Argument nur eine Form hat.

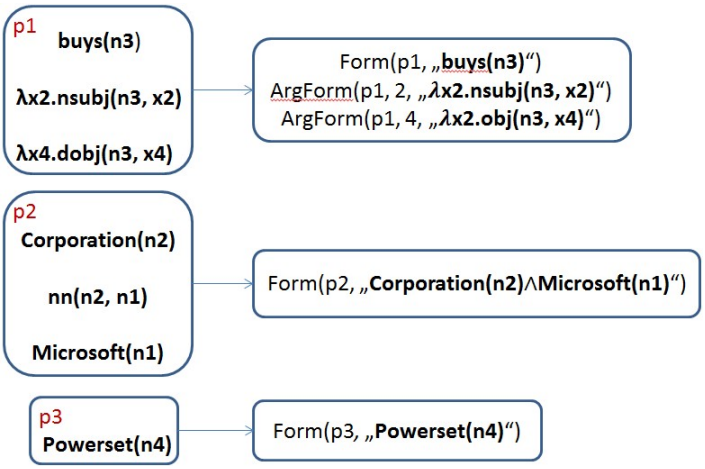


Abbildung 21 : Beispiel für QLF Partition. Form(p, f!), ArgForm(p, i, f!) sind QLF Partitionen.

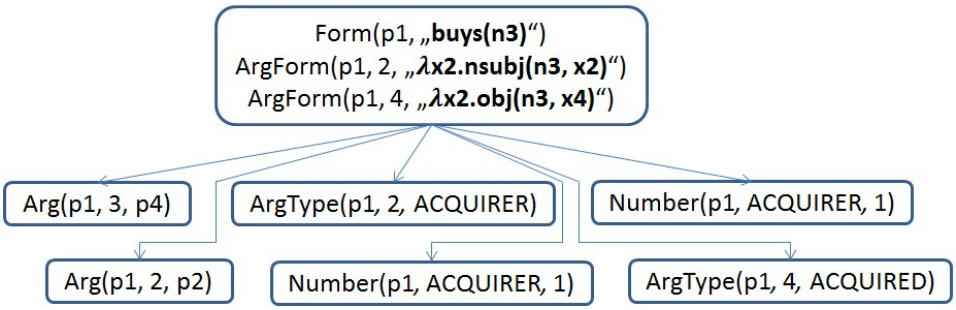


Abbildung 22 : Beispiel für ArgType(p, i, a!), Arg(p, i, p‘), Number(p, a, n)

Die o.g. semantische Grammatik Regeln von semantischer Analyse in USP werden durch folgende 4 Formeln definiert :

$$\mathbf{p \in +c \wedge Form(p,+f)}$$
$$\mathbf{ArgType(p, i, +a) \wedge ArgForm(p, i, +f)}$$
$$\mathbf{Arg(p, i, p') \wedge ArgType(p, i, +a) \wedge p' \in +c'}$$
$$\mathbf{Number(p, +a, +n)}$$

2.2.7.2 Ontology Unsupervised Semantic Parsing

Eine Schwachheit des USP Systems ist das „Sparse Data“, das „Sparse Data“ bezeichnet hier die Information, die selten im Korpus vorkommt. „„Sparse Data“ führt zu niedriger Genauigkeit, weil nicht genug Daten zur Verfügung stehen, um die Wahrscheinlichkeit des Ereignisses genau abzuschätzen.“ [26] Onto USP ist die Erweiterung von USP und hat die Fähigkeit, die Informationen zu strukturieren. Im Vergleich zu USP Onto USP führt eine Hierarchie Clusteranalyse durch. Onto USP löst das Problem über das „Sparse Data“ durch das Hierarchie Clusteranalyse.

In Abbildung 23 kann man erkennen, dass es eine Hierarchie zwischen den Clustern gibt. Ein „object cluster“ entspricht einer semantischen Bedeutung z.B. ACQUIRE und enthält ein oder einige „property cluster“ z.B. ACQUIRER. In „core form“ Cluster sind alle Variationen, die die gleiche Bedeutung besitzen. In „property cluster“ sind die Argumente von Variationen aus dem „core form“ Cluster. Mit der „IsPart“ Funktion wird die Hierarchie durch Verwendung einer Regel erstellt. Die s.g. Regel ist :

$$\mathbf{x \in +p \wedge HasValue(x, +v)}$$
$$\mathbf{e \in c \wedge SubExpr(x, e) \wedge x \in p \Rightarrow \exists^{1} i.IsPart(c, i, p)}$$

wobei :

- HasValue(s, v) : Sub-Form der Lambdaform hat dem Wert v.
- e ∈ c : in Cluster c gibt es Lambdaform e.
- SubExpr(s, e) : s ist eine Sub-Form einer Lambdaform.
- IsPart(c, i, p) : i-ste Eigenschaft Cluster p in Cluster c. Durch die Kombination der Sub-Formen der Lambdaformen erzeugt diese Funktion.

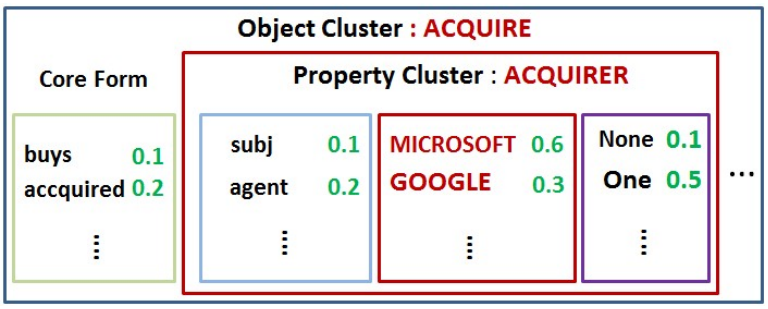


Abbildung 23: Beispiel für Objekt und Eigenschaft Cluster. In „property cluster“ sind Argumentformen, Argumente von Core Formen und Argument-Numbers.

3 Entwurf

Der natürliche Text wird von Stanford Parser und USP verarbeitet, die Ausgaben von USP System sind die „*.parse“ und „*.mln“ Dateien, wobei die „*.parse“ Datei die semantische Parser liefert und die „*.mln“ Datei einen Markov Logik Netzwerk liefert. Durch die Analyse von diesen zwei Dateien wird eine Hierarchie für die Wörter erstellt. Danach werden diese Wörter gefiltert und die Ausgaben von Produkten, Produkteigenschaften und die Werte der Produkteigenschaften generiert.

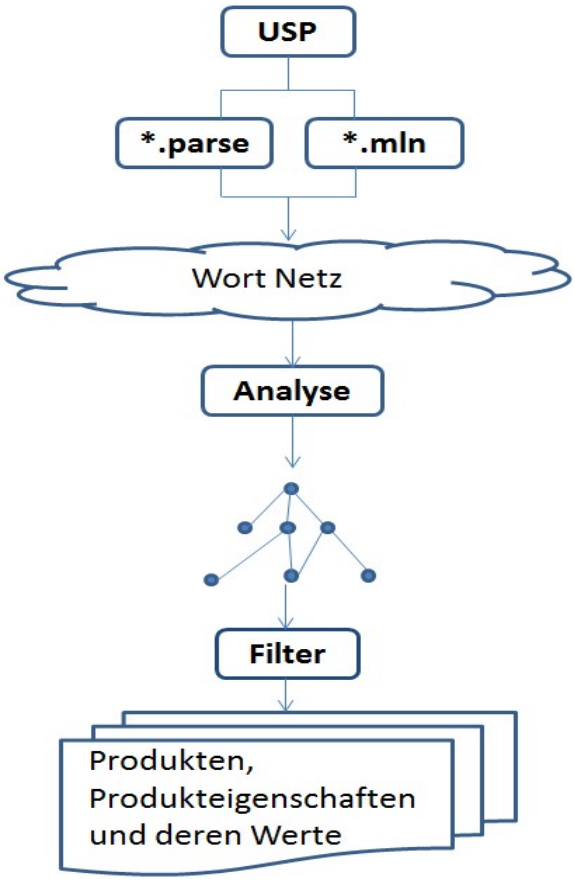


Abbildung 24: Architektur für Extraktion von Produkten, Produkteigenschaften und Produkteigenschaftswerten.

3.1 Ausgabedatei für USP

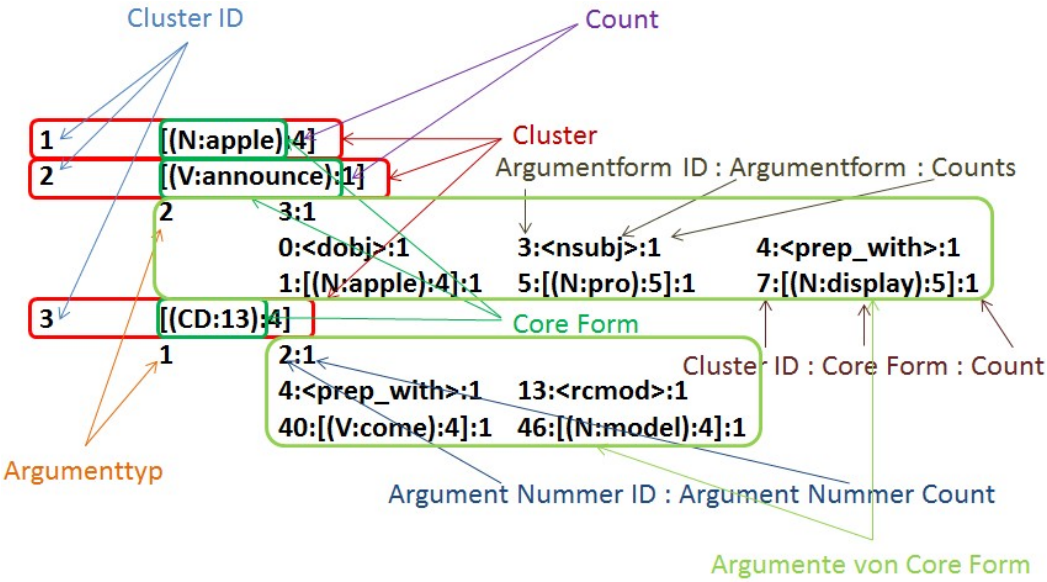


Abbildung 25 : Ein Beispiel für eine MLN Datei

In der MLN Datei sind die semantischen Parser. Jeder Cluster besitzt eine eindeutige Identifikation(Abk. ID), neben der Cluster ID ist eine Darstellung von Core Form. Eine Darstellung von Core Form besteht aus einem Worttyp, dem Wort von Core Form und der Anzahl(*Count*) von Core Form. Die Anzahl in der Darstellung von Core Form beschreibt, wie viel mal das Wort in dem Dokument vorkommt. In dem Programm wird diese Anzahl vom Cluster *ClusterSum* benannt. Unter der Zeile von Cluster ist die Beschreibung der Argumente von dem Cluster, wenn das Wort die Beziehungen mit anderen Wörtern hat. Die Beschreibung der Argumente von dem Cluster besitzt drei Zeilen, in der ersten Zeile ist der Argumenttyp, und anschließend sind die Argument Nummer Identifikation und Anzahl von Argument Nummer zu finden. In der zweiten Zeile ist die Beschreibung über Argumentform und die Beschreibung besteht aus einer eindeutigen Identifikation, Argumentform und Anzahl der Argumentform. In der dritten Zeile ist die Beschreibung über die benachbarten Wörter. Diese Beschreibung besteht aus einer eindeutige Cluster ID von Core Form Word, der Darstellung von Core Form und die Anzahl des Worts, und diese Anzahl beschreibt, wie viel mal das Wort als Nachbar vorkommt. Z.B. 40:[(V:come):4]:1 bedeutet : Core Form Word „come“ besitzt Cluster ID 40, hat den Typ „V“, kommt insgesamt 4 mal in dem Dokument vor und kommt 1 mal als Nachbar von Wort „13“.

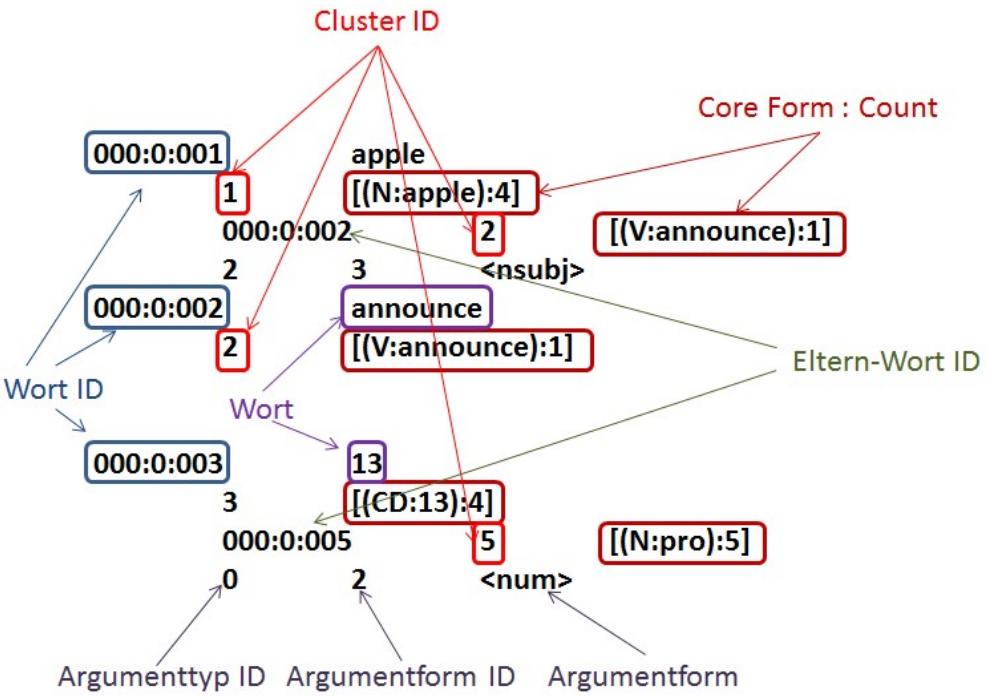


Abbildung 26 : Ein Beispiel für eine PARSE Datei

In der ersten Zeile der „parse“ Datei sind die Word ID und Wort zu finden. Eine Word Identifikation ist eindeutig für jedes Wort, und die Darstellung von Wort ID ist „DokumentID:AbschnittID:TokenID“. In der zweiten Zeile sind die eindeutige Cluster ID und die Darstellung von Core Form. In der dritten Zeile sind die eindeutige Eltern-Wort ID, eindeutige Cluster ID von Eltern-Wort und die Darstellung von Core Form. In der vierten Zeile sind die Argumenttyp ID, eindeutige Argumentform ID und Argumentform.

Die „parse“ Datei und „*.mln“ Datei implizieren die Strukturen von Wörtern, weil jede Cluster-ID in der „*.mln“ Datei mit einer oder mehren Cluster-ID durch eine oder mehre Beziehungen verbindet, und jede Wort-ID in der „parse“ Datei mit einer eindeutigen Eltern-Wort-ID verbindet. Mit Hilfe der Software „GVEdit“ wird eine „mln“ Datei als ein Wort Netz visualisiert. Jeder Knoten entspricht einem Wort aus Core Form, jede Kante repräsentiert eine Beziehung zwischen den zwei Wörtern und entspricht einer Argumentform. Folgende sind die Graphische Darstellung von „mln“ Datei und für einen kleinen Text mit zwei Abschnitten.

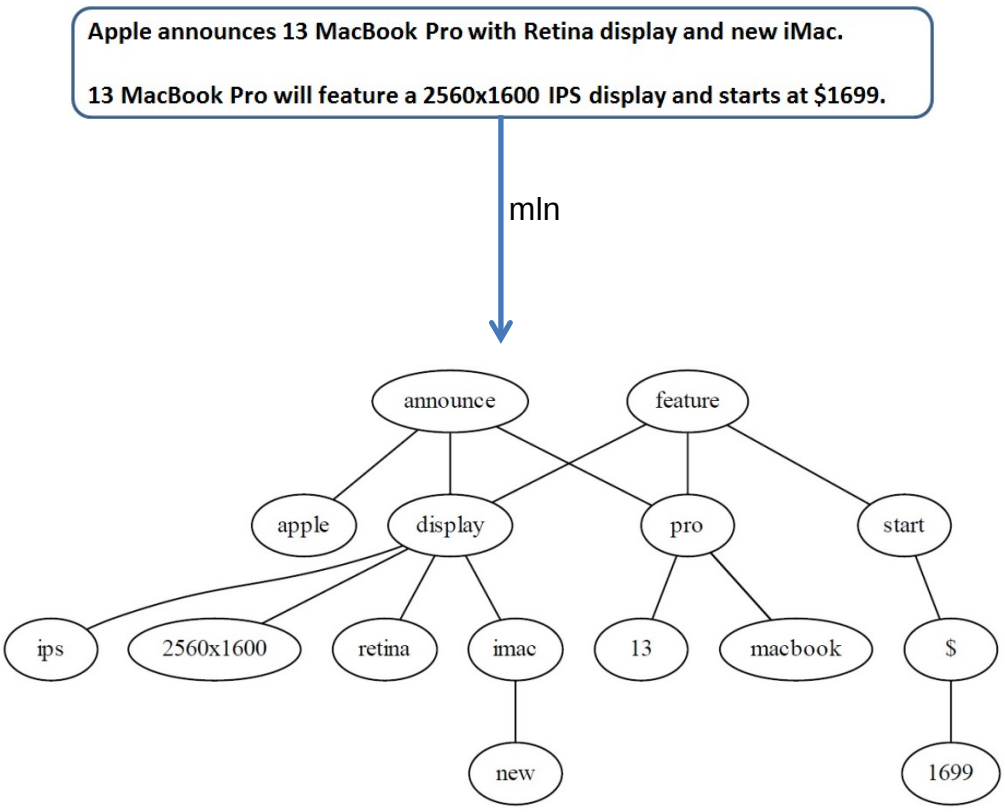


Abbildung 27 : Beispiel für die graphische Darstellung von „mln“ Datei

3.2 Der erste Ansatz : Generierung der Regeln für Informationsextraktion

Dieser Ansatz wurde von Betreuer Julian Eichhoff mit mir zusammen entwickelt. In [18] wählt man die sehr oft vorkommenden Daten aus, damit mit der Hilfe von PMI und „Naive Bayes Classifier“ die Regeln für die Informationsextraktion generiert werden. Die Idee ist ähnlich. Durch die Analyse der Beziehungen zwischen den Wörtern werden einige Regeln generiert, damit das Produkt und die Produkteigenschaft zuerst extrahiert werden können. Um die „mln“ Datei zu analysieren, habe ich die Software WEKA benutzt. WEKA(mehr sehen [27]) ist ein Data Mining Software und sammelt viele Maschinenlernen Algorithmus für die Aufgaben von Data Mining. Ich habe 300 Daten benutzt, um die Regeln für die Informationsextraktion zu finden. Die Annahmen sind :

- Die sehr oft vorkommenden, gefundenen Wörter sind Produkteigenschaft, weil verschiedene Produkte sehr möglich die gleiche Produkteigenschaft haben.
- Die Produkteigenschaft bindet direkt mit Produkt und Produkteigenschaftswert bindet direkt mit Produkteigenschaft
- Produkteigenschaft sind am meisten die Nomen
- Produkte sind Nomen
- Verb ist kein Produkt, keine Produkteigenschaft, kein Produkteigenschaftswert

Die Regeln werden zwar schon gefunden, aber man muss die folgende Probleme lösen :

- Nach der Annahme, die sehr oft vorkommenden, gefundenen Wörter sind Produkteigenschaft, s.g. „oft“ muss möglichst gut definiert werden.
- Wie bindet Produkteigenschaft mit Produkt? Wie bindet Produkteigenschaft mit Produkteigenschaftswert? D.h. wie ist die Hierarchie der Wörter?
- Die Regeln, die mit der Hilfe von WEKA generiert werden, gelten theoretisch nur für die Daten, die in WEKA eingesetzt werden.

Um die Regeln zu generieren, muss man die Dateien in ein Eingabeformat von WEKA umwandeln. Die Eingabe für „EKA ist „*.arff“ Datei. In einer „*.arff“ Datei werden die Instanzen und die Attribute der Instanzen beschrieben. D.h. eine „*.arff“ Datei stellt eine Datenmenge bereit. Die für ca. 300 Testdateien erzeugte „*.arff“ Datei ist ca. 200MB groß. Es könnte viele irrelevante oder redundante Informationen für das Lernverfahren geben. [28] Eine reduzierte „*.arff“ Datei ist ca. 2 MB und wird wieder als Eingabe für WEKA benutzt. Durch Analyse des Ergebnisses von WEKA werden die Regeln für Informationsextraktion gestellt :

- Produkteigenschaften sind am meisten Nomen.
- Produkt ist Norm.
- Wenn ein Wort Worttyp „CD“ hat, ist das Wort sehr möglich ein Produkteigenschaftswert.
- Wenn ein Wort Worttyp „V“ hat, ist das Wort kein Produkt, keine Produkteigenschaft oder kein Produkteigenschaftswert.
- Produkt und Produkteigenschaft sind in Verbindung mit Argumentform „amod“
- Produkt und Produkteigenschaft sind in Verbindung mit Argumentform „appos“
- Produkt und Produkteigenschaft sind in Verbindung mit Argumentform „num“
- Produkt und Produkteigenschaft sind in Verbindung mit Argumentform „nn“
- Produkt und Produkteigenschaft sind in Verbindung mit Einheiten

Nach der ersten Annahme sind die sehr oft vorkommenden gefundenen Wörter sind Produkteigenschaft. Aber es ist immer schwierig zu lösen, wie kann man „sehr oft“ definieren. Sei „sehr oft“ so dass, in jedem Text mindestens einmal der Begriff vorkommt. Viele Eigenschaften werden ignoriert, weil nicht alle Produkteigenschaften in jedem Text vorkommen. Sei „sehr oft“ für 300 Dateien 100, dann werden viele irrelevante Wörter extrahiert.

Zwar werden die Regeln generiert, aber das Problem für die Unterscheidung zwischen Produkt und Produkteigenschaft immer noch nicht gelöst. Das Problem geht darauf wieder zurück, eine Hierarchie aufzubauen.

3.3 Der zweite Ansatz : Generierung der Hierarchie der Wörter

Ein natürlicher Text wird mit Hilfe von USP bzw. OntoUSP in Markov Logik Netzwerk für die Wörter umgewandelt. Ein MLN verknüpft Prädikatenlogik erster Stufe und Markov-Netze miteinander. Deshalb kann das Wissen mittels der Regeln aus dem vorhandenen Wissen Schlüsse inferieren. Eine Hierarchie der Wörter kann als die Abhängigkeiten zwischen den Wörtern gesehen werden. Die Erstellung der Hierarchie kann als ein Problem für Bestimmung der Abhängigkeiten zwischen den Wörtern gesehen werden. Die von USP System erzeugte „*.mln“ Datei liefert einen Markov Logik Netzwerk bzw. ein Netz der Wörter. Die Idee geht davon aus, dass durch die Analyse von „*.mln“ Datei bzw. von dem Netz der Wörter die Abhängigkeiten bestimmt werden.

In Abbildung 11 wird die Ursache von Krebs beschreibt: wenn man raucht, leidet man an Krebs, d.h. Rauchen führt zu Krebs. Die formale Darstellung für die Aussage „Rauchen führt zu Krebs“ ist ∀xSm(x) ⇒ Ca(x). Durch die Lambda Reduktion wird die Form gebildet : Sm(A) ⇒ Ca(A). Die graphische Darstellungen sind :

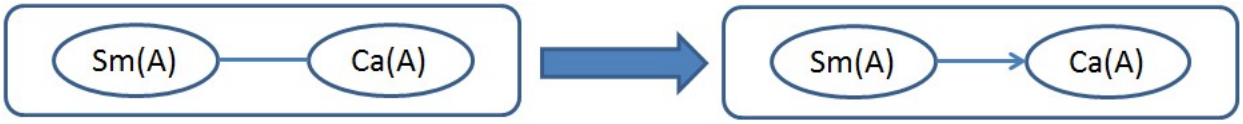


Abbildung 28 : links ist die graphische Darstellung von Markov Logik Netzwerk. Rechts ist die gerichtete graphische Darstellung, entspricht der graphische Darstellung von MLN.

Für jedes Paar von zwei Knoten entspricht die graphischen Darstellung von MLN einer gerichteten graphischen Darstellung. Der Elternknoten repräsentiert Ursache und der Kindknoten repräsentiert die Wirkung. Der Kindknoten wird aus den Elternknoten abgeleitet. Mit anderen Worten, das Vorkommen der Kindknoten ist abhängig von dem Vorkommen der Elternknoten. Das Problem von Extraktion von Produkten, Produkteigenschaften und Produkteigenschaftswerten ist das Problem von Bestimmen der Abhängigkeit zwischen den Knoten.

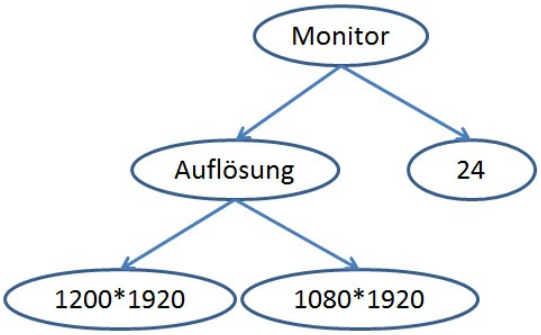


Abbildung 29 : Ein weiteres Beispiel für graphische Darstellung von „*.mln“ Datei

Ein gerichteter Graph besitzt auch eine Hierarchie. Elternknoten liegt in der höhen Ebene und Kindknoten liegt eine Ebene niedriger. Eine Hierarchie ist illustriert in Abbildung 29. In dieser Hierarchie kann der Elternknoten und Kindknoten in einer gleichen Klasse sein, und auch in verschiedene Klasse sein. Aber die Klasse, in der die Kindknoten liegen, ist nicht höher als die Klasse, in der die Elternknoten liegen. Z.B. „24“ und „Monitor“ gehören zu einer gleichen Klasse, trotzdem ist „24“ ein Kindknoten.

Um die Idee zu erklären, werde ich die Definitionen und Annahme wiederholen :

- Ein Blatt ist ein Knoten, der keine Kindknoten hat.
- Ein Wurzel Knoten ist der Knoten, der keine Elternknoten hat.
- Die Blätter sind die Produkteigenschaftswerte.
- Die Produkteigenschaftswerte verbinden sich nicht mit Verb Wörtern.
- Die Elternknoten der Blätter sind Produkteigenschaften.
- Die Wurzel ist entweder ein Produkt oder ein Markenzeichen.
- Ein Produkt ist ein Nomen. Verb Wort ist kein Produkt, keine Produkteigenschaft, kein Produkteigenschaftswert.

Die Idee ist :

- Bestimmen der Abhängigkeit zwischen den Blätter und ihren Eltern Knoten.
- Nach der Annahme werden die Blätter in die Menge vom Produkteigenschaftswert hinzugefügt, und die direkten Elternknoten der Blätter werden in die Menge von Produkteigenschaft hinzugefügt.
- Bestimmen der Abhängigkeit zwischen den Knoten.
- Während der Bestimmung der Abhängigkeiten wird die Hierarchie gleichzeitig erstellt.
- Nach der Annahme werden die Wurzel Knoten in die Menge von Produkt hinzugefügt.
- Nach der Annahme werden alle nicht Normen Wort werden gefiltert.

Im Vergleich zu den vorherigen Ansätzen, z.B. [8], [10], [11], [12] und [13], Die Extraktion von Produkten, Produkteigenschaften und Produkteigenschaftswerten ist voll automatisch :

- Nur die Software für die Sprachanalyse ist erforderlich.
- Die Daten der Eingabe muss nicht mehr manuell markiert werden.
- Die Ontologie bzw. die Hierarchie der Wörter wird bei Analyse vom Netz der Wörter erstellt.
- Dieses Text Mining Verfahren gilt für natürlichen Text und offene Domänen.

4 Implementierung

4.1 Generierung der Hierarchie durch Bestimmung der Abhängigkeit von den Wörtern

Die Hauptaufgabe dieses Ansatzes ist die Bestimmung der Abhängigkeit zwischen dem Knoten, damit eine Hierarchie der Wörter erstellt wird. Danach wird die Hierarchie verbessert, schließlich werden die bestimmten Knoten in dem Graph gefiltert.

Einige Definitionen :

- Wenn ein Knoten *Count* und *ClusterSum* hat, kommt dieser Knoten aus dem Bereich „Argumente von Core Form“ (siehe in der Abbildung 30). „apple“, „pro“ und „display“ haben *Count* und *ClusterSum*. „announce“ hat *Clustersum* aber kein „*Count*“.
- Cluster ID von Knoten aus dem Bereich „Argumente von Core Form“ wird *CClusterID* genannt, *ClusterSum* von Knoten aus dem Bereich „Argumente von Core Form“ wird *CClusterSum* genannt.
- CC = *Count*/*CClusterSum*

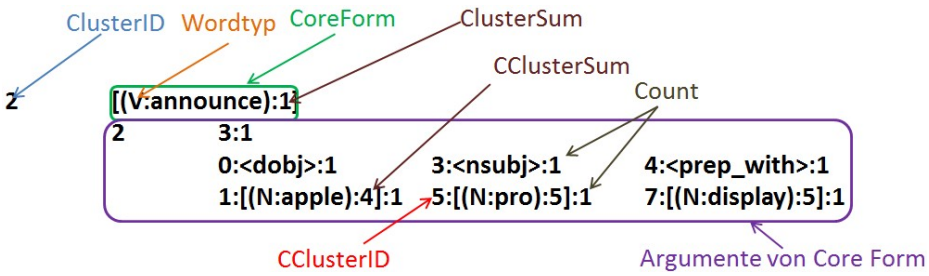


Abbildung 30 : Illustration für einige Definitionen

4.2 Vorverarbeitung

In diesem Schritt wird die Abhängigkeit zwischen den Blättern und Elternknoten bestimmt. Wenn die Hierarchie zwischen den Blättern und Elternknoten festgelegt wird, können die Produkteigenschaften und die Produkteigenschaftswerte nach der Annahmen in „*property*“ Menge hinzugefügt werden. Die Werte von *CC* werden für alle Knoten berechnet. Um die Hierarchie in dem nächsten Schritt weiter zu erstellen, müssen die Werte von *Count* und *CClusterSum* von Kindknoten und *ClusterSum* von Elternknoten aktualisiert werden.

4.2.1 Verschiedene Fälle

Um die Abhängigkeit zwischen den Knoten zu bestimmen, werden die folgende Situationen betrachtet :

- Viele Knoten kommen nur einmal im Graph vor. Solche Knoten kommen im Graph vor, genau dann wenn ihre Nachbarknoten im Graph vorkommen. Es gibt eine Ursachlichkeit bzw. Abhängigkeit dazwischen.
- Manche Knoten kommen einige Male im Graph vor. Die Knoten kommen im Graph vor, genau dann wenn ihre Nachbarknoten im Graphen vorkommen. Es gibt eine Ursachlichkeit bzw. Abhängigkeit dazwischen.
- Manche Knoten kommen einige Male im Graph vor. Der Knoten kommt im Graph am meisten zusammen mit ihrem Nachbarknoten vor. Es gibt eine Ursachlichkeit bzw. Abhängigkeit dazwischen.

Abbildung 31 illustriert die o.g. Fälle :

- „1920x1080“ kommt vor, genau dann wenn „display“ vorkommt.
- „retina“ kommt vor, genau dann wenn „display“ vorkommt.
- wenn „ips“ vorkommt, kommt „display“ am meisten vor.

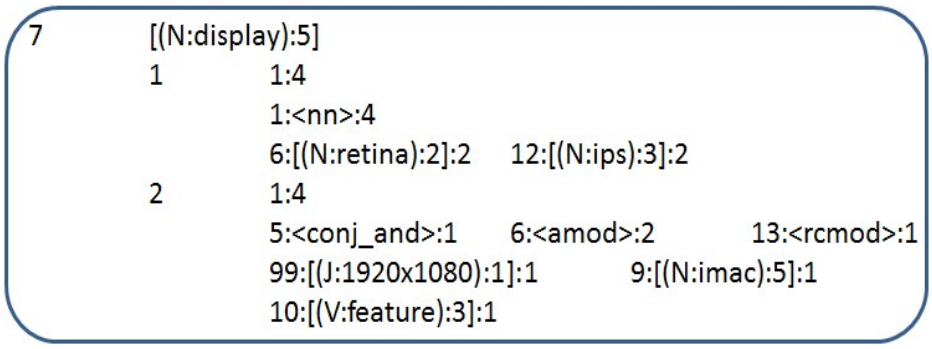


Abbildung 31 : Ein Teil von „*.mln“

4.2.2 Bestimmung der Abhängigkeit

CClusterSum und *Count* von „1920x1080“ sind 1. Das bedeutet :

- „1920x1080“ kommt nur einmal in diesem Text vor. „1920x1080“ kommt einmal als Nachbarknoten von „display“ vor. Das Vorkommen von „1920x1080“ ist nur abhängig von dem Vorkommen von „display“.

CClusterSum und *Count* von „retina“ sind 2. Das bedeutet :

- „retina“ kommt zweimal in diesem Text vor. „retina“ kommt zweimal als Nachbarknoten von „display“ vor. Das Vorkommen von „retina“ ist nur abhängig von dem Vorkommen von „display“.

Count von „ips“ ist 2 und *CClusterSum* von „ips“ ist 3. Das bedeutet :

- „ips“ kommt sehr oft zusammen mit dem Nachbarknoten „display“ vor. (Das Problem ist hier, wie man „oft“ möglichst gut definieren kann. Das ist eine Arbeit für die Zukunft. Hier wird die „oft“ als „*Count*/*ClusterSum* >=0.5” definiert)

Verallgemeinerte Bedeutung :

- *CClusterSum* = *Count* ⇒ Es gibt eine starke Abhängigkeit zwischen den Knoten. Wenn die Knoten aus dem Bereich „Argumente von Core Form“ im Text vorkommen, genau dann wenn ihre Nachbarknoten im Text vorkommen.
- *CC*>= 0.5 ⇒ Es gibt eine leichte Abhängigkeit zwischen den Knoten, die Abhängigkeit ist noch nicht deutlich.

Nach o.g. Aussage wird die Hierarchie der Knoten bestimmt. Der Knoten mit *CC*=1, ist ein Kindknoten von seinem Nachbarknoten. Abbildung 32 illustriert die Erstellung einer Hierarchie durch die Bestimmung der Abhängigkeit zwischen den Knoten. Nach der Bestimmung der Abhängigkeiten entstehen die Blätter in dem Graph. Der rote Knoten ist der Kindknoten und die Knoten „display“ ist Elternknoten von Kindknoten.

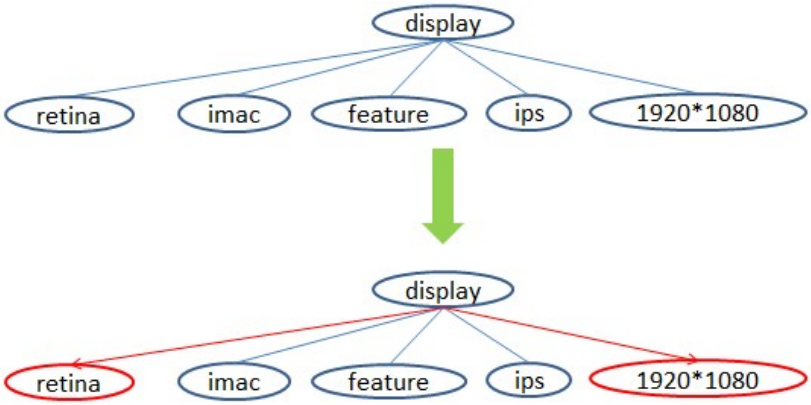


Abbildung 32 : Oben ist die graphische Darstellung von Abbildung 31. Nach der Bestimmung der Abhängigkeiten (*CC*=1) wird die graphische Darstellung (unten) erzeugt.

Nach o.g. Aussage wird der Knoten mit *CC*>=0.5 ausgewählt, weil es eine Abhängigkeit zwischen den Knoten und ihren Nachbarknoten gibt. Aber die Abhängigkeit kann in dem Schritt von Vorverarbeitung nicht bestimmt werden. In Abbildung 33 hat der grüne Knoten eine Abhängigkeit mit „display“, aber die Abhängigkeit ist unklar. Die Knoten „imac“ und „feature“ sind die Nachbarknoten von „display“, es gibt jetzt noch keine Abhängigkeit dazwischen.

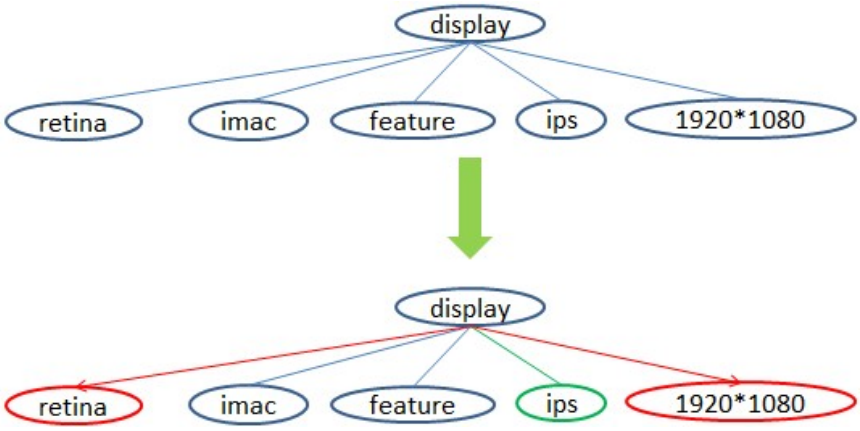


Abbildung 33 : Es gibt die Abhängigkeit zwischen Knoten „ips“ und „display“, aber die Abhängigkeit ist nicht deutlich

4.2.3 Bestimmung von Produkteigenschaften und Produkteigenschaftswerten

- Nach der Annahme sind alle Knoten, die Worttyp „V“ besitzt, keine Produkte, keine Produkteigenschaften und keine Produkteigenschaftswerte.
- Nach der Annahme sind die Blätter die Produkteigenschaftswerten
- Nach der Annahme sind die Elternknoten der Blätter Produkteigenschaften
- Nach der Annahme allen Knoten, die mit Verb Wörtern sich verbinden, sind keine Produkteigenschaftswerte.
- Wenn ein Cluster keine Kindcluster besitzt, z.B. „31“ in der Abbildung 34, sind die entsprechenden Cluster Knoten die Blätter.

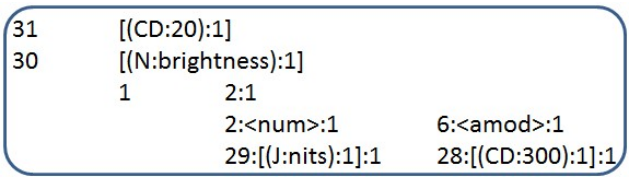


Abbildung 34 : Beispiel für die Darstellung von Blättern in „*.mln“, 31 ist ein Blatt und 30 ist kein Blatt.

Die gefundenen Knoten werden durch o.g. Annahmen gefiltert, schließlich werden die übrigen Knoten in einer „*property*“ Menge hinzugefügt. Um die Produkteigenschaften und Produkteigenschaftswerte in einer Menge zu unterscheiden, wird jedem Knoten ein Wert zugewiesen, Produkteigenschaft entspricht den Wert 0 und Produkteigenschaftswert entspricht Wert -10.

4.2.4 Aktualisierung der Werte von ClusterSum, Count und CClusterSum

Wenn ein Knoten als Kindknoten oder Elternknoten benannt ist, bedeutet dies das, die Abhängigkeit zwischen den Knoten und seinem Nachbarknoten bereit bestimmt ist.

- Der Knoten hat nur ein Blatt ⇒ *ClusterSum* = *ClusterSum* – *Count*, *CClusterSum*=0 und *Count*=0.
- Der Knoten besitzt mehrere Knoten ⇒
    - *searchNode* ist eine Funktion(aus java Code) und gibt eine Menge von WordID zurück. Die WordID kommt aus der „*.parse“ Datei (sieh Abbildung 26). Ein Knoten entspricht einem Wort, ein Wort kann mehre WordID haben, weil ein Wort in einem Text einige Male vorkommt. Alle WordID von einem Elternknoten mit einem bestimmten Kindknoten können durch *searchNote* bestimmt werden und in einer Menge, die „*processednode*“ benannt, hinzugefügt werden. Mit Hilfe von *searchNode* werden die WordID der Elternknoten von allen Kindknoten berechnet. Die Menge „*processednode*“ enthaltet alle WordID von Elternknoten, die die bestimmte Abhängigkeit mit Kindknoten haben.

- *ClusterSum*=*Clustersum* - |*processednode*| (Größe der Menge „*processednode*“).
- *Count* von jedem Kindknoten wird 0, d.h. *Count*=0.
- *CClusterSum* von jedem Kindknoten wird 0, d.h. *CClusterSum*=0

Sei WordID Menge vom Wort „display“ {000:0:003, 000:0:050}. Seien *searchNode*(retina, display)={000:0:050}, *searchNode*(ips, display)={000:0:003, 000:0:050} und *searchNode*(1920*1080, display)={000:0:050}, daraus folgt *processednode*={000:0:003, 000:0:050} und |*processednode*|=2.

	ClusterSum	CClusterSum	Count
display	5		
retina		2	2
1920*1080		1	1
ips		3	2

	ClusterSum	CClusterSum	Count
display	3		
retina		0	0
1920*1080		0	0
ips		1	0

Abbildung 35 : Ein Beispiel für die Aktualisierung der Werte von *ClusterSum*, *Count* und *CClusterSum*

```
searchNode (cid1, cid2) {
        while (all cid1' of mln) {
                if (cid1=cid1' && cid2=cid2'){
                        add WordID to WordIDSet
                }
        }
        return WordIDSet
}
```

Abbildung 36 : Pseudocode für *searchNode*. Wenn ein Wort A mit Cluster ID *cid1* ein Eltern-Word B mit Cluster ID *cid2* hätte, dann wird die WordID von Knoten B in einer Menge *WordIDSet* hinzugefügt. Schließlich gibt die Menge *WordIDSet* zurück.

In diesem Schritt werden alle Abhängigkeiten von den Knoten mit *CC*=1 bestimmt, und die Produkteigenschaften und Produkteigenschaftswerte gefunden. Die Knoten mit *CC*>=0.5 werden auch ausgewählt, es gibt die Abhängigkeiten zwischen diesen Knoten und ihren Nachbarknoten. Um die nicht deutlichen Abhängigkeiten zu bestimmen, müssen die Werte von *ClusterSum*, *Count* und *CClusterSum* aktualisiert werden.

4.3 Bearbeitung der Knoten mit CC>=0.5

Nach der Aktualisierung der Werte von *ClusterSum*, *Count* und *CClusterSum* entstehen 4 Fälle, durch die Analyse von diesen 4 Fällen werden die Abhängigkeiten bestimmt.

4.3.1 Vier Fälle

4.3.1.1 Der erste Fall

CClusterSum=0 und *Count*=0 Bedingung (1)

Sei *initClusterSum* ein niemals aktualisierte *ClusterSum*, d.h. der Wert kommt aus „*.mln“, *initCClusterSum* sei ein niemals aktualisierte *CClusterSum*. d.h. der Wert kommt aus „*.mln“. Die Abhängigkeit wird dadurch bestimmt :

- Solcher Fall tritt auf, wenn ein Knoten zu mehren Nachbarknoten gehört. Der Knoten, der die Bedingung (1) erfüllt, ist ein Kindknoten von ihrem Nachbarknoten, d.h. Nachbarknoten ist Elternknoten, z.B. „2560*1600“ in der Abbildung 37 und 38.
- Aber wenn *initClusterSum* = *Count*, ist die Abhängigkeit zwischen den Knoten anders. Sei Knoten A besitzt einen *initClusterSum* Wert, und Knoten B besitzt einen *Count* Wert. Die Bedingung „*initClusterSum* = *Count*“ bedeutet, dass Knoten A im Text insgesamt n-mal vorkommt, und sein Nachbarknoten B auch n-mal vorkommt. Knoten B kann mehr als n-mal im Text vorkommen. Mit anderen Worten, der Knoten A kommt in einem Text vor, genau dann wenn Knoten B vorkommt, deshalb ist Knoten B ein Elternknoten von seinem Nachbarknoten A, und sein Nachbarknoten A ist Kindknoten, z.B. „imac“ in der Abbildung 37 und 38.

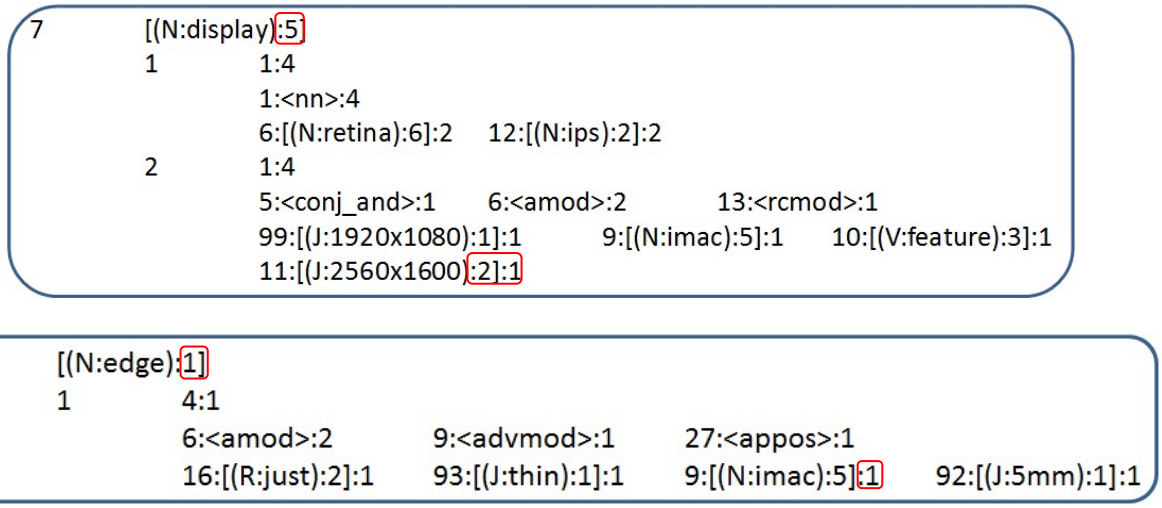


Abbildung 37 : „*.mln“ für den 1. Fall

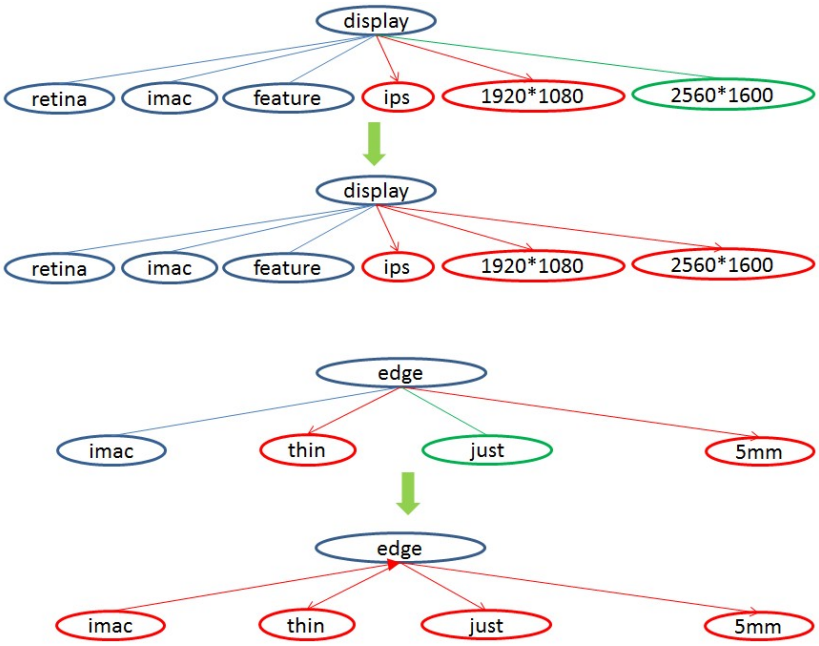


Abbildung 38 : Ein Beispiel für den 1. Fall

4.3.1.2 Der zweite Fall

CClusterSum>0 und *Count*=0 Bedingung (2)

Wenn ein Knoten, der die Bedingung (2) erfüllt, im Text vorkommt, kommt ihr Nachbarknoten auch vor.

```
5       [(N:pro):5
        0       2:5
                1:<nn>:7  2:<num>:3
                35:[(N:generation):2]:1         3:[(CD:13):4]:3         4:[(N:macbook):6]:5
                6:[(N:retina):6]:1
        2       1:2
                6:<amod>:2
                34:[(J:previous):2]:1           8:[(J:new):4]:1
        4       2:1
                4:<prep_with>:1     12:<poss>:1
                21:[(PRP$:they):2]:1            7:[(N:display):5]:1
```

Abbildung 39 : „*.mln“ für 2.Fall

Der Knoten „3:[(CD:13):4]:3“ und der Knoten „4:[(N:macbook):6]:5” erfüllen die Bedingung (2), aber die Abhängigkeiten zwischen der Knoten „[(N:pro):5]“ und der Knoten „3:[(CD:13):4]:3“ und der Knoten „4:[(N:macbook):6]:5” sind Unterschiedlich :

- Der Knoten „3:[(CD:13):4]:3” kommt im Text am meisten zusammen mit Knoten „[(N:pro):5]“ vor. D.h. der Knoten „3:[(CD:13):4]:3“ zu mehren Nachbarknoten gehört.
- Der Knoten „[(N:pro):5]“ kommt im Text vor genau dann wenn der Knoten „4:[(N:macbook):6]:5” vorkommt. Mit anderen Wort, es gibt eine starke Abhängigkeit

zwischen der Knoten „4:[(N:macbook):6]:5” und der Knoten „[(N:pro):5]“, d.h. „4:[(N:macbook):6]:5” ist ein Elternknoten von „[(N:pro):5]“, und „[(N:pro):5]“ ist ein Kindknoten.

Sei *initClusterSum* eine niemals aktualisierte *ClusterSum*, *intCClusterSum* eine niemals aktualisierte *CClusterSum*. Die Abhängigkeit wird dadurch bestimmt :

- Wenn ein Knoten die Bedingung (2) erfüllt und *initClusterSum*>= *initCClusterSum*, dann ist dieser Knoten ein Kindknoten von Nachbarknoten.
- Wenn ein Knoten Bedingung (2) erfüllt und *initClusterSum*<*initCClusterSum*, dann ist dieser Knoten ein Elternknoten von Nachbarknoten.

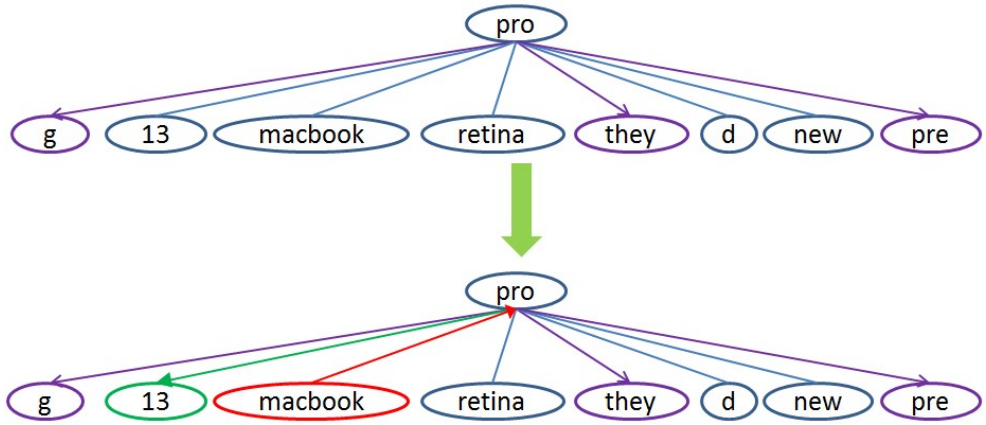


Abbildung 40 : Ein Beispiel für 2.Fall

4.3.1.3 Der dritte Fall

CClusterSum=0 und *Count*>0 Bedingung (3)

Wenn ein Knoten, der die Bedingung (3) erfüllt, im Text vorkommt, kommt ihr Nachbarknoten auch vor.

```
33      [(J:thinner):2]
        0       1:2
                14:<npadvmod>:2
                32:[(N:%):1]:1      95:[(CD:80):1]:1
        1       1:2
                15:<prep_than>:2
                35:[(N:generation):2]:1      5:[(N:pro):5]:1
```

Abbildung 41 : „*.mln“ für 3.Fall

Abbildung 42 : Ein Beispiel für 3.Fall

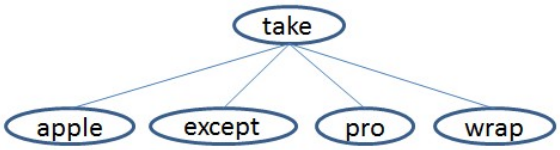
4.3.1.4 Der vierte Fall

CClusterSum>0 und *Count*>0 Bedingung (4)

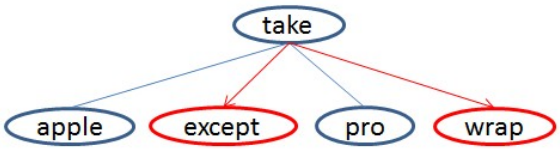
Wenn ein Knoten A nur einen Nachbarknoten B, der die Bedingung (4) erfüllt, hat, dann ist Knoten A ein Kindknoten von Knoten B und B ist ein Elternknoten. Wenn ein Knoten mehrere Nachbarknoten hat, welche die Bedingung (4) erfüllen können, wird dieser Knoten im nächsten Schritt weiter verarbeitet.

```
19      [(V:take):1]
        3       4:1
                0:<dobj>:1      3:<nsubj>:1     8:<advcl>:1     11:<prep_off>:1
                1:[(N:apple):4]:1   18:[(V:expect):1]:1  20:[(N:wrap):1]:1   5:[(N:pro):5]:1
```

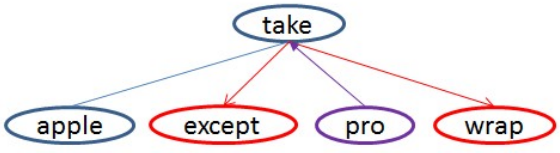
Abbildung 43 : „*.mln“ für 4.Fall



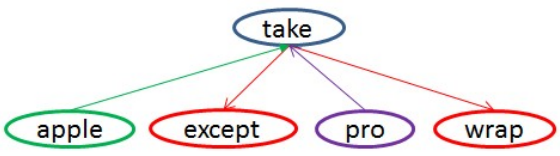
	ClusterSum	CClusterSum	Count
take	1		
apple		4	1
except		1	1
pro		5	1
wrap		1	1



	ClusterSum	CClusterSum	Count
take	0		
apple		4	1
except		0	0
pro		5	1
wrap		0	0



	ClusterSum	CClusterSum	Count
take	0		
apple		4	1
except		0	0
pro		0	1
wrap		0	0



	ClusterSum	CClusterSum	Count
take	0		
apple		4	1
except		0	0
pro		0	0
wrap		0	0

Abbildung 44 : Ein Beispiel für die Bestimmung der Abhängigkeiten. Links ist die Graphische Darstellung für ein Cluster, rechts ist eine Tabelle für *ClusterSum*, *ClusterSum* und *Count*.

In der Abbildung 44, die Abhängigkeiten zwischen „take“ und „except“ sowie „wrap“ werden in Vorverarbeitungsschritt bestimmt, die Abhängigkeiten zwischen „take“ und „pro“ wird nach der Bedingung (3) bestimmt, und die Abhängigkeiten zwischen „take“ und „apple“ wird nach der Bedingung (4) bestimmt.

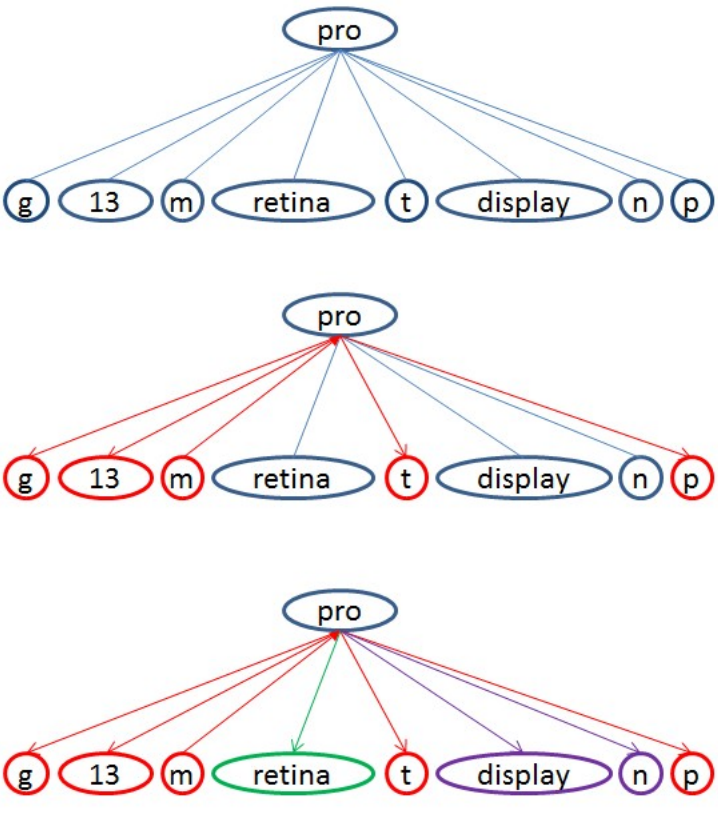
4.3.2 Aktualisierung der Werte von ClusterSum, Count und CClusterSum

Der Verfahren der Aktualisierung der Werte von *ClusterSum*, *Count* und *CClusterSum* ist gleich wie der Fall „Der Knoten besitzt mehrere Knoten“ in 4.2.4.

4.4 Verarbeitung der Konten mit *CC*<0.5

4.4.1 Bestimmung der Abhängigkeit

Bis jetzt kann die Abhängigkeit zwischen den Knoten und ihren Nachbarknoten mit *CC*>=0.5 bestimmt werden. Es gibt einige Knoten mit *CC*<0.5. Die Idee für die Verarbeitung von dem Knoten mit *CC*<0.5 ist : man aktualisiert die Werte von *Count*, *CClusterSum* und *ClusterSum* bis zu *CC* Wert>=0.5, damit die vorherige Ansätze wieder verwendbar sind.



	ClusterSum	CClusterSum	Count
pro	5		
generation		2	1
13		4	3
macbook		6	5
retina		6	1
previous		2	1
new		4	1
they		2	1
display		5	1

	ClusterSum	CClusterSum	Count
pro	0		
generation		0	0
13		1	0
macbook		1	0
retina		4	1
previous		0	0
new		2	1
they		0	0
display		2	1

	ClusterSum	CClusterSum	Count
pro	0		
generation		0	0
13		0	0
macbook		0	0
retina		2	0
previous		0	0
new		0	0
they		0	0
display		0	0

Abbildung 45 : ein Beispiel für die Bestimmung der Abhängigkeiten. Links ist die Graphische Darstellung für ein Cluster, rechts ist eine Tabelle für *ClusterSum*, *ClusterSum* und *Count*.

„display“, „new“ und „retina“ besitzt den Wert *CC*>0.5, durch maximal zwei Aktualisierung der Werten *ClusterSum*, *ClusterSum* und *Count* werden die *CC* Werte bis zu kleiner gleich 0.5 reduziert.

Schließlich gibt es keine Knoten mit *CC*<0.5. Wenn es keine Knoten mit *CC*<0.5 gibt, wird die Hierarchie der Knoten erstellt. Ein ungerichteter Graph in einen gerichteten Graph umgewandelt.

Das Verfahren von der Aktualisierung der Werte von *ClusterSum*, *Count* und *CClusterSum* ist gleich wie der Fall „Der Knoten besitzt mehre Knoten“ in 4.2.4.

4.4.2 Aktualisierung der Werte von ClusterSum, Count und CClusterSum

Im Vorverarbeitungsschritt werden die *CC* Werte berechnet. Wenn der *CC* Wert größer gleich 0.5 ist, dann werden entsprechende Knoten in *mnm*(Merged Node Map) hinzugefügt, die aktualisierten Werte von *ClusterSum* werden in *mergedmap_c2s* hinzugefügt und die aktualisierter Werte von *CClusterSum* und *Count* werden in *mergedmap_c2cnsc* hinzugefügt.

Einige Definitionen :

- *mnm* : *LinkedHashMap* ClusterID = [ClusterID, die entsprechende Knoten mit CC>=0.5]
- *mergedmap_c2s : LinkedHashMap* ClusterID = neuer *ClusterSum*
- *mergedmap_c2cnsc* : *LinkedHashMap* ClusterID = [ClusterID!neuer *ClusterSum*!neue *Count*]

Aktualisierung der Werte von *ClusterSum*, *Count* und C*ClusterSum* für die Knoten mit CC≠1 werden dadurch erlegdigt, *mnm*, *mergedmap_c2cnsc* und *mergedmap_c2cnsc* zu aktualisieren.

- Geht von *mergedmap_c2*s aus. Wenn CC Wert größer gleich 0.5, dann wird entsprechende Knoten in *mnm* hinzugefügt. Bemerkung : Die Bedingung (n2sum != 0 && n2count != 0) implementiert 4. Fall in 4.2.2.1. Nur die Knoten, die 4. Fall erfüllt, werden verarbeitet, weil die Abhängigkeiten von anderen Knoten schon festgelegt werden.

```
while (mergedmap_c2cnsc is not empty)
{
        if (Count/CClusterSum>=0.5) {
          add ClusterID into mnm
         }
}
```

Abbildung 46 : Pseudocode für Berechnung von CC Wert und *mnm*

- Geht von *mergedmap_c2s* aus, wenn der Knoten kein Blatt ist, wird die Menge von „*processednode*“ mithilfe der Funktion *searchNode* berechnet. Die Größe der Menge

von „*processednode*“ beschreibt, wie viele Abhängigkeiten zwischen diesem Knoten und anderen Knoten schon bestimmt werden. Die originale *ClusterSum* minus Anzahl von festgelegten Abhängigkeiten ist neue *ClusterSum* bzw. *CClusterSum*.

```
while (mergedmap_c2s is not empty) {
        if (node is processed) {
            new CClusterSum = CClusterSum - the number of processednode
            add new CClusterSum into mergedmap_c2s
            }
}
```

Abbildung 47 : Pseudocode für Berechnung von neuer *ClusterSum*

- Geht von *mergedmap_c2cnsc* aus, neu *Count* ist alter *Count* Wert minus die Anzahl der festgelegten Abhängigkeiten. Nur die Knoten, die 4. Fall erfüllt, werden verarbeitet, weil die Abhängigkeiten von anderen Knoten schon festgelegt werden.

```
while (mergedmap_c2cnsc is not empty) {
        new Count = Count – searchNode (cid2, cid)
}
```

Abbildung 48 : Pseudocode für Berechnung von neuem *Count*

4.5 Verbessern die Hierarchie

Zwar werden die Abhängigkeiten bis jetzt schon bestimmt, aber es gibt immer noch einige Fehler. Mit folgenden Funktionen werden die Fehler korrigiert.

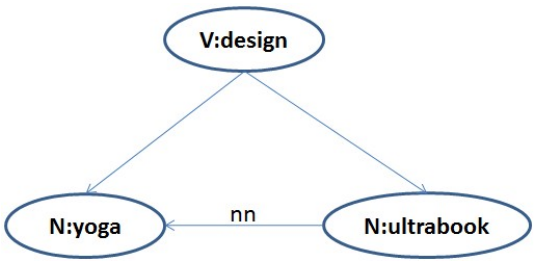


Abbildung 49 : Beispiel für eine falsche Bestimmung der Abhängigkeit zwischen den Knoten

- Wenn ein Verb Wort die Abhängigkeiten zwischen den Normen hat und eine Abhängigkeit „nn“ zwischen den Normen besteht. Dann werden die Abhängigkeit zwischen Verb und Norm, die als einen Kindknoten repräsentiert wird, gelöscht. Der Grund für die Korrektur der Abhängigkeit ist : „nn“ Abhängigkeit ist stärker als die Abhängigkeit zwischen Verb und Norm. Um die Korrektur zu realisieren, muss zuerst das Wort Verb gefunden werden. Danach werden alle Kindknoten von Verb durchgelesen, um die die Beziehung zwischen den Kindknoten festzulegen. Der Kindknoten A sei Elternknoten von Kindknoten B. Wenn die Beziehung zwischen den

Knoten A und Knoten B „nn“ ist, wird die Anhängigkeit zwischen Verb und Knoten B gelöscht.

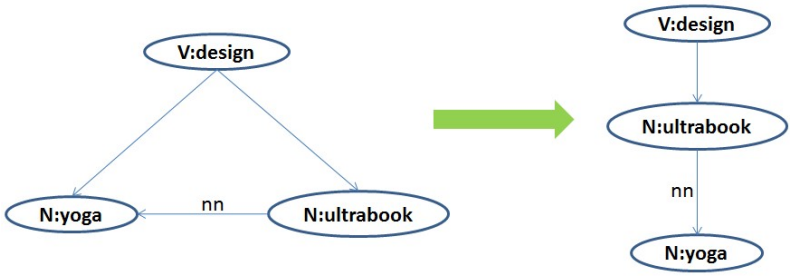


Abbildung 50 : Beispiel für eine Korrektur der Abhängigkeit

- Wenn die Knoten die in Abbildung 51 dargestellte Beziehung haben und Knoten A hat keine Beziehungen mit anderen Knoten, wird die Abhängigkeit zwischen den Knoten korrigiert. Diese Abbildung 51 illustriert einen solchen Fall.

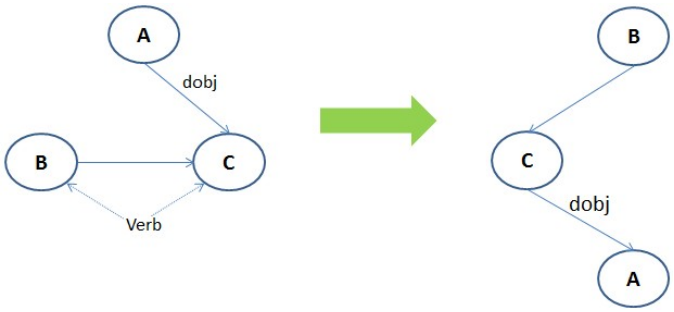


Abbildung 51 : Beispiel für Korrektur der Abhängigkeit

Um die Korrektur zu realisieren, muss zuerst den Knoten A gefunden werden. Danach werden die Argumentform von Knoten A bestimmt. Alle Elternknoten von Knoten C werden durchgelesen. Wenn ein Elternknoten von Knoten C ein Verb ist, korrigiert das Programm die Abhängigkeit von „A->C“ in „C->A“. Die Gründe der Korrektur sind : A ist ein Objekt und gehört zu einem Subjekt. Ein Elternknoten hat Einfluss auf die Kindknoten, aber die Elternknoten von dem Elternknoten keinen Einfluss auf die Kindknoten. Das heißt, wenn die Abhängigkeit zwischen Knoten A und C sich ändern, gibt es keinen Einfluss auf die Abhängigkeit zwischen Knoten B und seine Nachbarknoten.

- Wenn die Knoten die in Abbildung 52 dargestellte Beziehung haben, dann wird die Abhängigkeit korrigiert. Knoten V1 ist ein Kindknoten von Knoten root und Knoten root hat keine anderen Nachbarknoten, soll Knoten V3 keinen Einfluss auf Knoten root haben. Mit anderem Wort, das Vorkommen von Knoten root ist abhängig von dem Vorkommen von Knoten V1. Abbildung 52 illustriert die Korrektur.

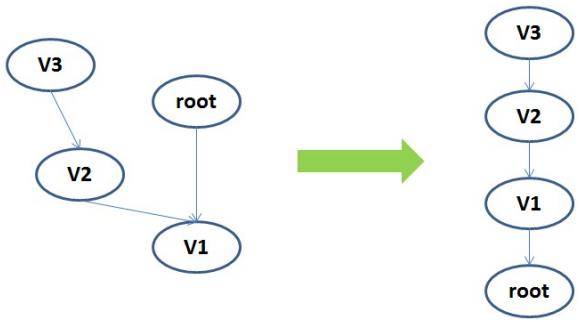


Abbildung 52 : Beispiel für Korrektur der Abhängigkeit

Nun sind die Abhängigkeiten zwischen den Knoten festgelegt. Aber es könnte jedoch noch Zyklen auftreten. Eine Hierarchie muss ein Baum oder ein gerichteter azyklischer Graph sein. Daher müssen mögliche Zyklen entfernt werden. Bindet ein Verb zwei Knoten, die Abhängigkeit bereits bestimmt wurde, führt dies zu einer Zyklus. Um den Zyklus zu entfernen, muss zuerst der Zyklus gefunden werden, danach wird die Abhängigkeit zwischen dem Verb und seinem Kindknoten entfernt. Abbildung 53 53 illustriert die Korrektur.

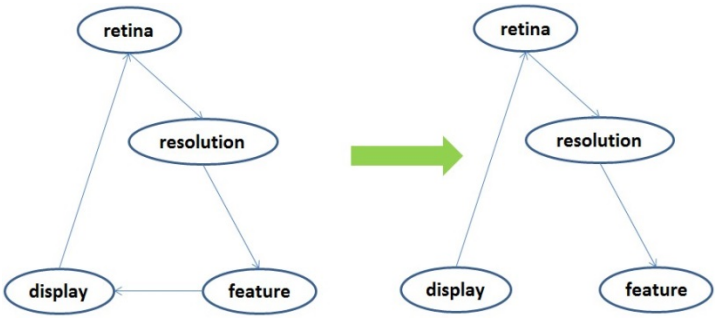


Abbildung 54: Beispiel für die Entfernung eines Zyklus

Durch o.g. Verfahren wird die Hierarchie korrigiert. Um eine bessere Hierarchie zu erhalten, ist eine Erweiterung erforderlich. Einige Regeln, die in 3.3 generiert werden, kann man zum Korrektur der Abhängigkeit benutzten. Aus meiner Erfahrung ist die Korrektur der Abhängigkeit mit einzelnen Regeln, ist es möglich, dass die richtige Hierarchie falsch korrigiert wird.

4.6 Filter

Alle gefundene Produkt, Produkteigenschaft und Produkteigenschaftswerte werden noch einmal gefiltert, um die Klassifikation der Wörter zu verbessern.

- Einige Wörter, die als Produkteigenschaftswerte erkannt werden, sind keine Produkteigenschaftswerte. Diese Wörter in der Hierarchie sind Kindknoten von Produkteigenschaften, aber sie haben die Beziehung zwischen den diesen Kindknoten

und Produkteigenschaften, d.h. die Argumentform ist „nn“. Wenn die Wörter die Argumentform „nn“ haben, sind die Wörter eine Nominalphrase. Das heißt, wenn die Elternknoten eine Produkteigenschaft ist, ist die Kindknoten nicht ein Produkteigenschaftswert sondern eine Produkteigenschaft, weil die Nominalphrase aus diesen Wörtern besteht. Es geht von Eigenschaftswert in „*property*“ Menge aus, wenn das Wort des Eigenschaftswerts mit seinem Elternknoten die Argumentform „nn“ hat, dann wird das Wort des Eigenschaftswerts als Eigenschaft korrigiert. Abbildung 54 illustriert das Verfahre von Filter.

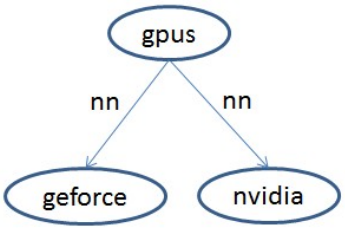


Abbildung 55 : Beispiel für den Filter

- Wenn die Beziehung, ausgehend von der „root“ Menge, zwischen einer Wurzel und seinem Kindknoten „nn“ ist und sein Kindknoten ein Kindknoten von einer anderen Wurzel ist, gehört dieser Kindknoten gehört zu anderem Produkt. Abbildung 55 illustriert das Verfahre von Filter.

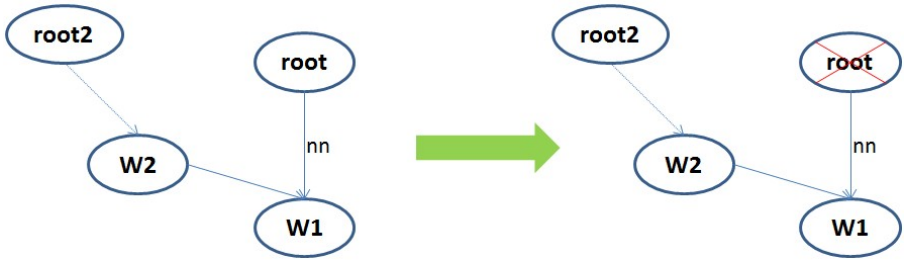


Abbildung 56 : Beispiel für den Filter

- Sei ein Knoten A ein Kindknoten von einer Wurzel B. Im Pfad von Knoten A bis zur Wurzel C gibt es mehrere Verben. D.h. Knoten B ist ein Kindknoten von einer anderen Wurzel, d.h. dieser Kindknoten gehört zu einem anderen Produkt. Abbildung 56 illustriert das Verfahre des Filterns.

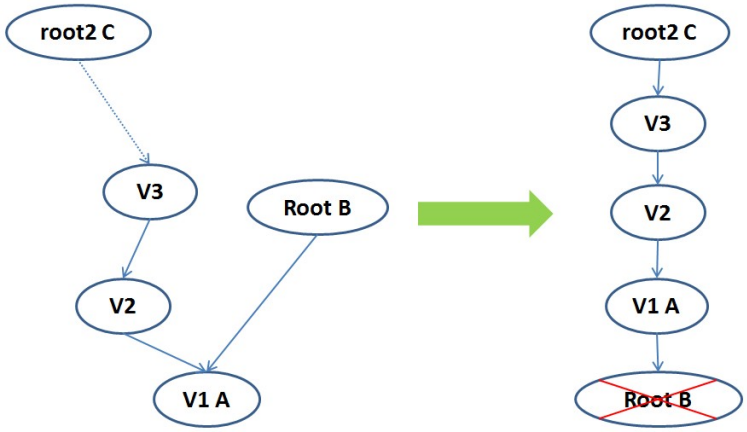


Abbildung 57 : Beispiel für den Filter

- Einige Wörter, die als Produkte erkannt werden, sind keine Produkte. Diese Wörter in der Hierarchie sind Wurzel Knoten, haben aber eine Beziehung „conj_and“ oder „conj_or“ zwischen den Wörtern und ihre Kindknoten(d.h. die Argumentform ist „conj_and“ oder „conj_or“). Wenn ihre Kindknoten Produkteigenschaften sind, ist es möglich, dass die Wurzel Knoten ebenfalls Produkteigenschaft sind. Wenn ein Kindknoten einer Wurzel, ausgehend von einer Wurzel Menge aus, eine Produkteigenschaft ist, dann wird diese Wurzel als Produkteigenschaft bezeichnet. Abbildung 57 illustriert das Verfahre des Filterns.

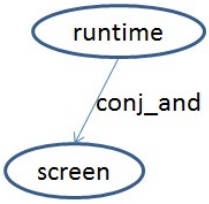


Abbildung 58 : Beispiel für den Filter

Der Filter hat keinen Einfluss auf die Änderung der Hierarchie, deshalb hat die Korrektheit der Hierarchie einen großen Einfluss auf die Filter. Wenn das Produkt, Produkteigenschaft und Produkteigenschaftswert nicht richtig erkannt werden, werden die Ergebnisse von Filter auch nicht exakt.

4.7 Ausgaben Filter

Einige Wurzel Knoten keine Normen sind. Nach der Annahme in 3.3 werden alle Wörter, die keine Normen sind, gefiltert.

5 Experiment

39 Daten werden zum Testen benutzt. Diese Texte kommen aus einem Website. Diese Texte sind über elektronische Geräte, z.B. Tablets, Notebook, HIFI. Die System Anforderung von USP ist sehr hoch, deshalb werden die kurze Texts ausgewählt. In einer Hierarchie der Wurzel ist eine Kategorie, wie Wurzel „Tablet“ ist Kategorie von „Windows 8 Tablet“. Wenn eine Kategorie von einem Produkt finden, dann wird dieses Produkt richtig gefunden. Wie in Abbildung 58 gezeigt, wenn Knoten A als Produkt erkennt, dann wird das Produkt A gefunden. Wie ein Produkt richtig darstellt wird, ist eine Aufgabe für die Zukunft.

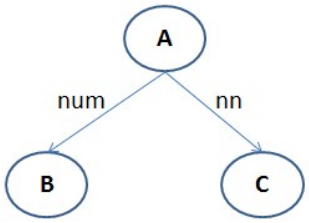


Abbildung 59 : Eine Darstellung für ein Produkt, sei der Produktname „BAC“, „A“ repräsentiert eine Kategorie

Die Genauigkeit von Produkt, Produkteigenschaft und Produkteigenschaftswert werden in der folgenden Abbildungen gegeben. Die Genauigkeit beschreibt den Anteil von relevanten Wörtern an der Menge von gefundenen Wörtern :

$$P = \frac{\text{relevante Wörter}}{\text{gefundene Wörter}}$$

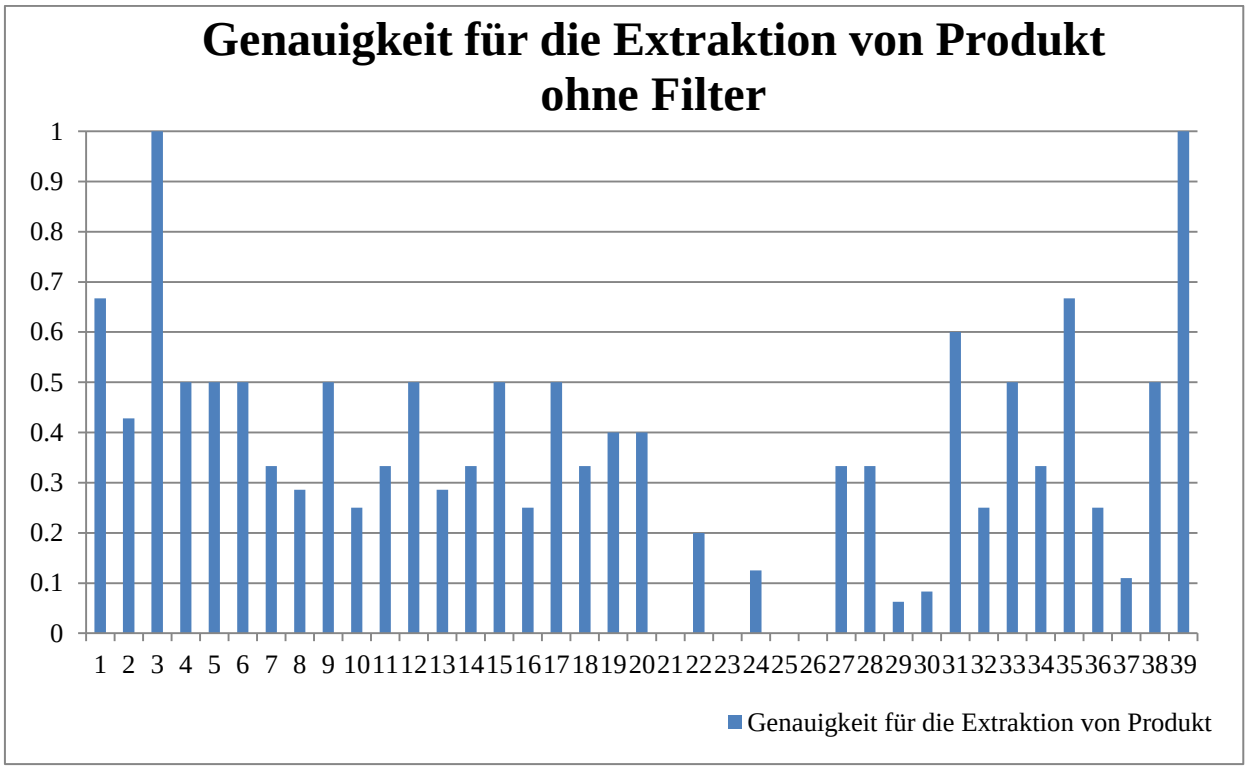


Abbildung 60 : Genauigkeit für die Extraktion von Produkt ohne Filter, X-Achse: die Nummer von Text Data, Y-Achse : die Genauigkeiten.

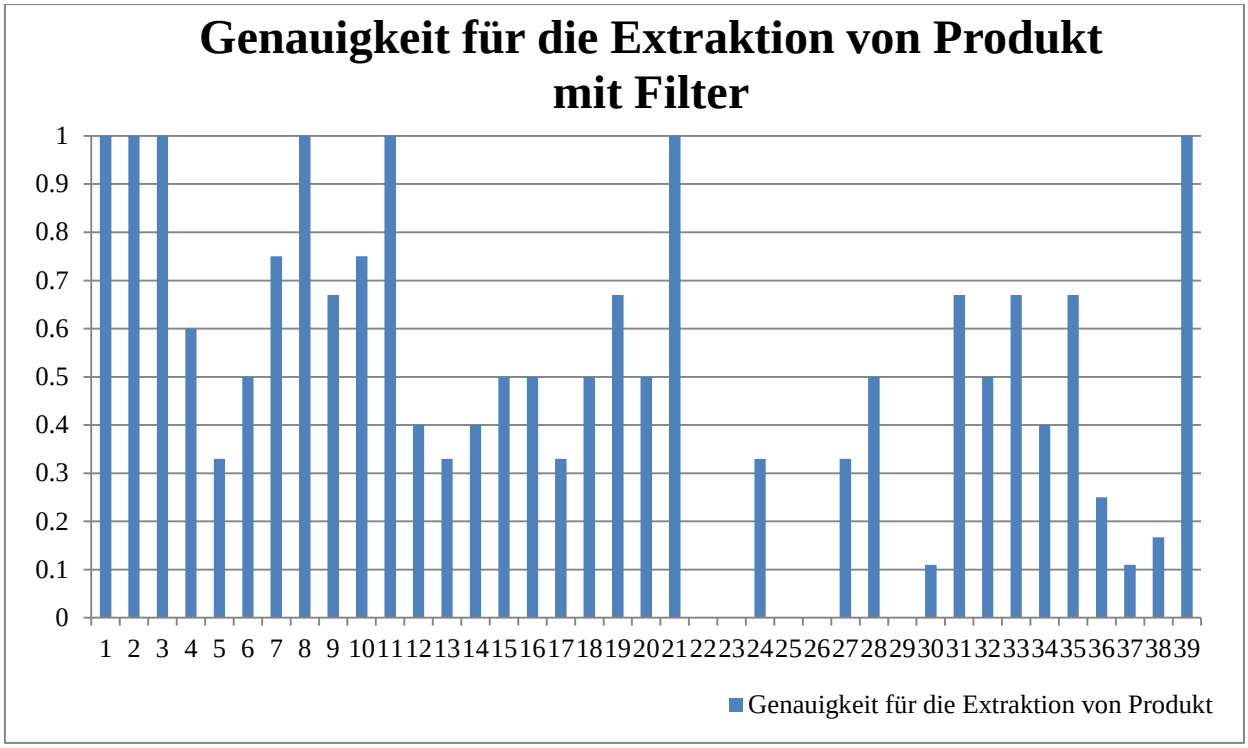


Abbildung 61 : Genauigkeit für die Extraktion von Produkt mit Filter. X-Achse: die Nummer von Text Data, Y-Achse : die Genauigkeiten.

In Abbildung 59 ist die Genauigkeit für die Extraktion von Produkten ohne Filter dargestellt. Die durchschnittliche Genauigkeit beträgt ca. 0.3627. In der Abbildung 60 ist die Genauigkeit für die Extraktion von Produkt mit Filter dargestellt. Die durchschnittliche Genauigkeit beträgt ca. 0.4984. Durch die Verwendung von verschiedenen Filtern wird die Genauigkeit auf ca. 37.4% erhöht. Aus den zwei Abbildungen kann man erkennen, dass ein guter Filter die Genauigkeit erhöhen kann. Jedoch ist es auch möglich, dass gebrauchte Wörter gefiltert werden, z.B. : Test Data 22 und Test Data 29. In dem Programm wird eine Abfolge der Randbedingungen benutzt. Ein „Voting Algorithmus“ wäre ein Vorschlag für zukünftige Arbeiten.

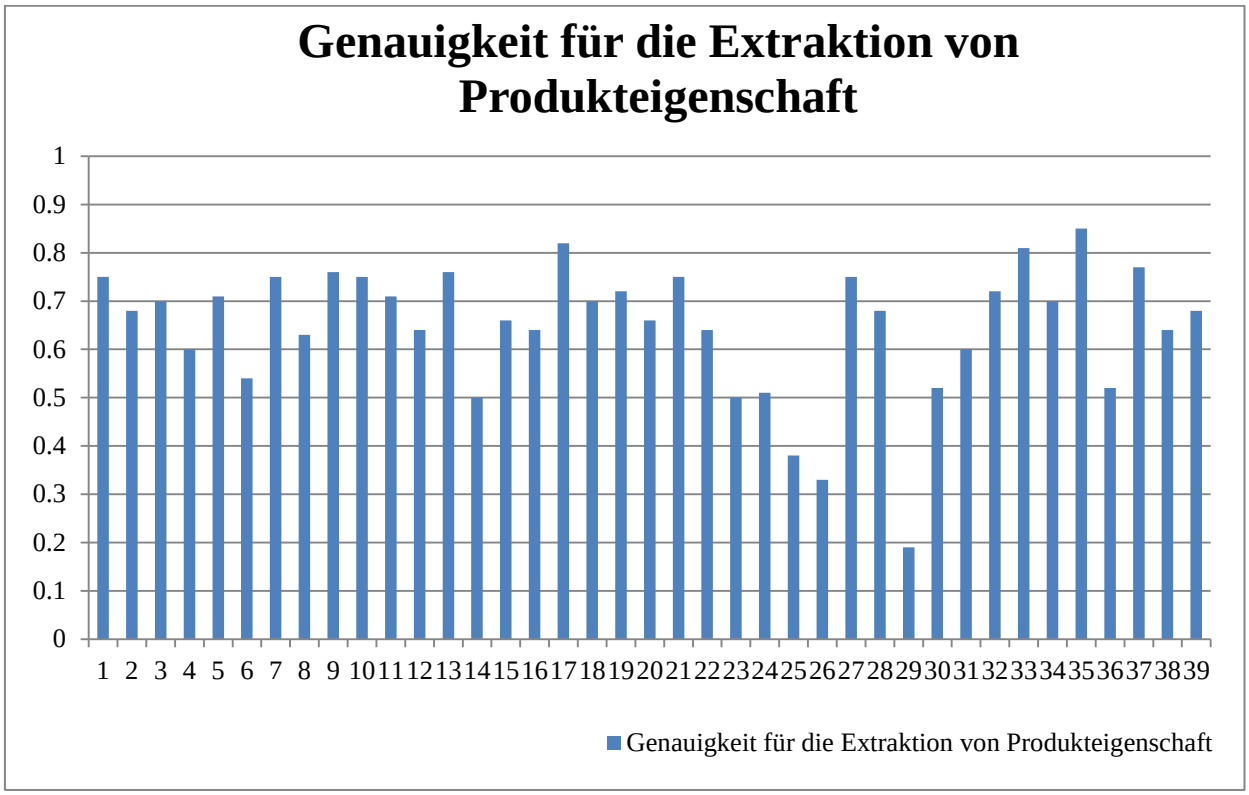


Abbildung 62 : Genauigkeit für die Extraktion von Produkteigenschaft. X-Achse: die Nummer von Text Data, Y-Achse : die Genauigkeiten.

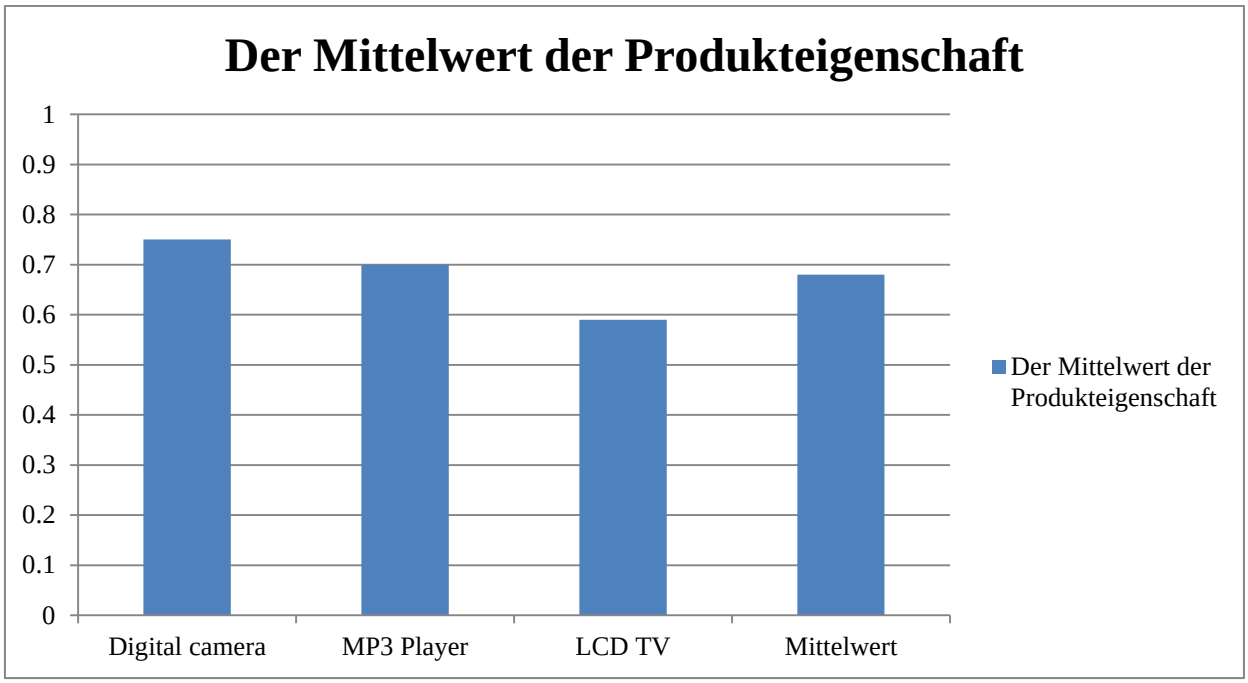


Abbildung 63 (aus [29]): Genauigkeit für die Extraktion von Produkteigenschaft. X-Achse: sind die Typen von Text Data, Y-Achse: die Genauigkeiten.

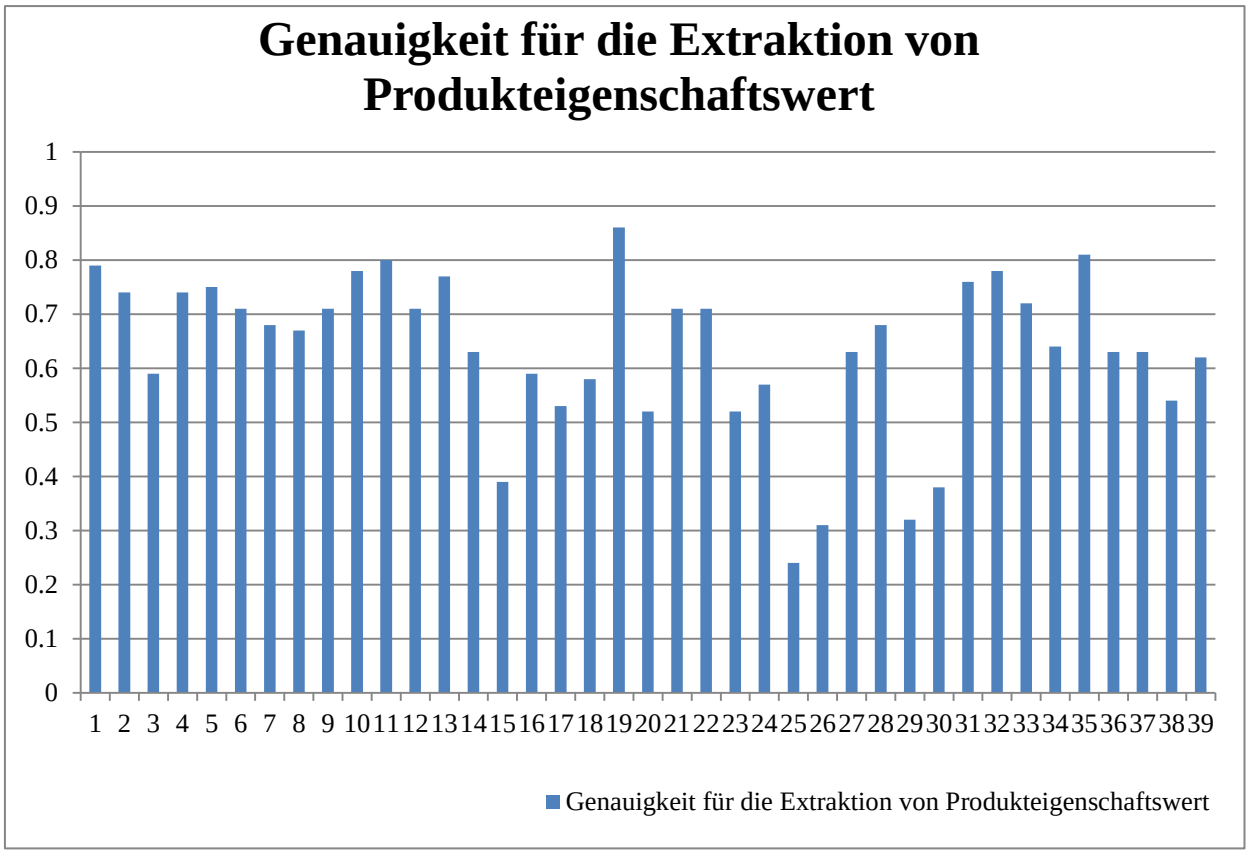


Abbildung 64 : Genauigkeit für die Extraktion von Produkteigenschaftswert. X-Achse: die Nummer von Text Data, Y-Achse : die Genauigkeiten

Der Mittelwert der Genauigkeit der Produkteigenschaft ist ca. 0.6467 und der Mittelwert der Genauigkeit Produkteigenschaftswert ist ca. 0.6344. Die beiden Genauigkeiten sind nicht sehr hoch, weil einerseits viele irrelevante Wörter nicht gefiltert werden, anderseits die Wörter nicht vollständig klassifiziert werden. In [29] wird ein nicht-überwachte Verfahren für die Normalisierung der Produkteigenschaften vorgestellt. Im Vergleich zu dem Ergebnis aus [29], ist der Mittelwert der Genauigkeit der Extraktion von Produkteigenschaften ca. 0.68. Zwar ist der Mittelwert der Genauigkeit nicht höher als der Mittelwert der Genauigkeit in [29], aber das Verfahren in dieser Diplomarbeit ist voll automatisch und unabhängig von dem Format des Text, mithilfe der Erweiterung ist die Verbesserung der Genauigkeit ebenfalls möglich.

Aus den Ergebnissen des Experiments kann die Schlussfolgerungen gezogen werden. Die Extraktion von den Informationen über die Produkte aus einem natürlichen Text ist sehr komplex, weil es in einem natürlichen Text viele irrelevante Wörter über die Produkte gibt. Während der Erstellung der Hierarchie werden die irrelevanten Informationen in dem Programm nicht gefiltert. Einige irrelevante Informationen werden mit Filtern gefiltert. Aber es gibt immer noch einige irrelevante Informationen, welche den stärkeren Einfluss auf den Genauigkeiten für die Extraktionen von Produkteigenschaften und Produkteigenschaftswerten haben. Wenn die Hierarchie eines Texts sehr gut aufgebaut wird, dann ist die Genauigkeit für die Extraktion von Produkt sehr hoch. Wenn die irrelevanten Informationen möglichst entfernt werden können, erhöht sich die Genauigkeit von Produkt, Produkteigenschaft und Produkteigenschaftswert. Wenn es in einem natürliche Text mehre exakte Informationen über das Produkt als allgemeine oder implizite Informationen gibt, ist die Genauigkeit von Produkteigenschaft und Produkteigenschaftswert höher. In dem Programm wird eine Abfolge der Randbedingungen benutzt. Ein „Voting“ Algorithmus entspannt die Randbedingungen, sodass es möglich ist, ein Wort zu filtern, wenn es eine Randbedingung in dem Programm verletzt.

6 Zusammenfassung

Ein natürlicher Text wird mit der Hilfe von USP bzw. OntoUSP in eine MLN für die Wörter umgewandelt. Ein MLN verknüpft Prädikatenlogik erster Stufe und Markov Netzwerk miteinander. Deshalb kann das Wissen mittels der Regeln aus dem vorhandenen Wissen Schlüsse inferieren. Die Korrektheit der Ergebnisse von „Standford Parse“ und USP hat starken Einfluss auf die Schlussfolgerung. Durch die Analyse von MLN kann eine Hierarchie für die Wörter erstellt werden. Die Hierarchie kann man verbessern. Zwar ist die Hierarchie nicht ideal, aber das Produkt, die Produkteigenschaft und der Produkteigenschaftswert können extrahiert werden. Im Vergleich zu anderen Ansätzen ist der Algorithmus, der in dieser Diplomarbeit entwickelt wird, voll automatisch. Durch die Verbesserung der Hierarchie werden die Genauigkeit der Informationsextraktion verbessert. Wenn die Hierarchie besser aufgebaut wird, werden die mehrere richtige Informationen extrahiert. Bei der Verbesserung der Hierarchie ist immer die Schwierigkeit, die Fehler aus den Ergebnissen von Standford Parse und USP System zu korrigieren. Durch die Korrektur der Abhängigkeit wird die Hierarchie verbessert. Die Korrektur ist meistens abhängig von syntaktischen Beziehungen, aber die richtige Korrektur muss mit einer Kombination von syntaktischen Beziehungen durchgeführt werden. Beim Filter werden die Ausgabe zu verbessern, führt keine mehr Korrektur von Abhängigkeit durch. Aus den Ergebnissen der Experimente kennt man das, wenn die Wörter, die mit dem Produkt irrelevant sind, gefiltert werden können oder die Hierarchie verbessert werden kann, wird die Genauigkeit verbessert.

7 Ausblick

Durch Erstellung einer Hierarchie der Wörter werden die Produkte, Produkteigenschaften und Produkteigenschaftswerte extrahiert, zwar ist die Genauigkeit nicht gut, aber es ist möglich zu erweitern.

7.1 Nicht löschbare Probleme

- Wie Abbildung 4 und Abbildung 5 zeigt, gibt es immer noch Probleme. Für eine richtige Erstellung einer Hierarchie der Wörter spielen Stanford Parse und USP System eine sehr wichtige Rolle. Wenn die Stanford Parse und USP System falsche Ergebnisse liefert, können die Fehler später nicht mehr korrigiert werden. In der Abbildung, „iMac“ ist eine Konstitution von „13 Macbook Pro“. Stanford Parse liefert kein richtigen Ergebnis. „iMac“ kann nicht als ein Produkt extrahiert werden, weil „iMac“ in „*.mln“ eine Konstitution von „display“ ist. Deswegen wird diese Fehler während der Bestimmung der Abhängigkeit nicht mehr korrigiert.

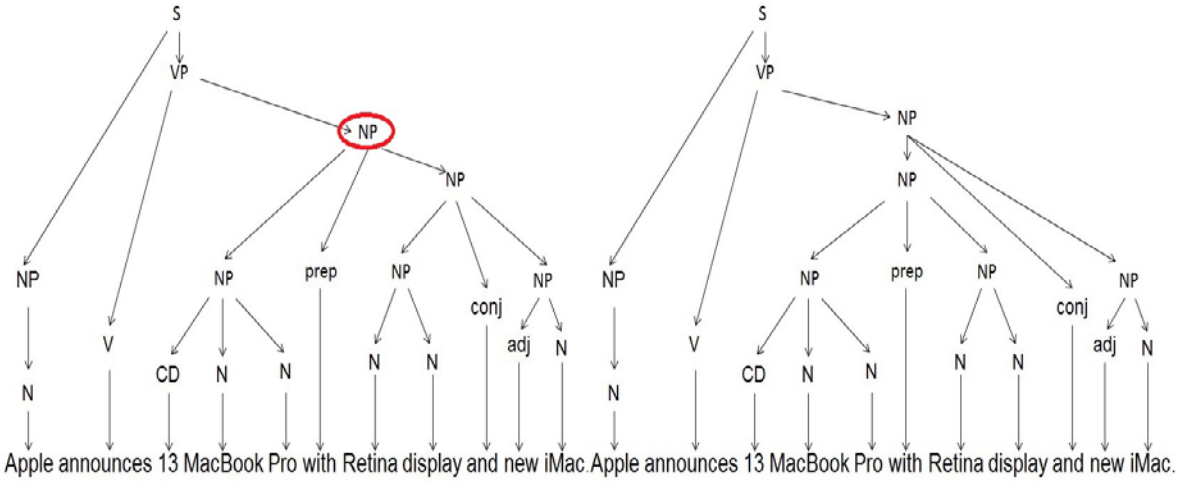


Abbildung 65 : Links ist die falsche Darstellung, rechts ist die richtige Darstellung.

- Wenn die Argumentform zwischen Kindknoten und Elternknoten „nn“ ist, dann ist die Abhängigkeit zwischen den Knoten schwierig zu bestimmen.

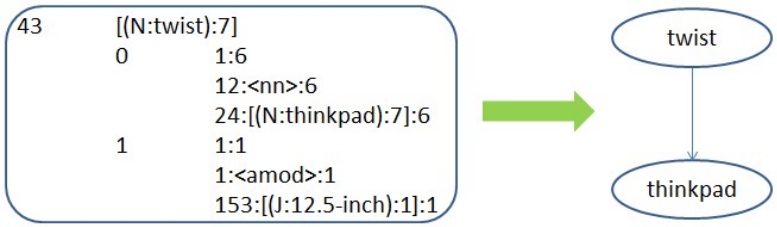


Abbildung 66 : Beispiel für zwei Wörter, die Argumentform „nn“ haben.

Im Beispiel „thinkpad“ ist ein Argument von Core Form „twist“. Tatsächlich ist „twist“ Serie von „thinkpad“, d.h. „thinkpad“ soll ein Elternknoten von „twist“ sein. Solche Fehler sind sehr schwierig zu korrigieren.

7.2 löschbare Probleme

Wie in „4.5 Verbessern die Hierarchie“ gezeigt, dass es einige löschbare Probleme gibt.

- Wenn zwei Wurzeln oder zwei Elternknoten die gleichen Produkteigenschaften haben, sind die zwei Knoten sehr möglich ein gleiches Produkt, z.B. „macbook“ und „device“.
- In dieser Diplomarbeit ist die Produkteigenschaft und Produkteigenschaftswert nur in dem Schritt Vorverarbeitung extrahiert. Basiert auf den Knoten, die gefundene Produkteigenschaft und Produkteigenschaftswert sind, können ihre Nachbarknoten weiter unterschieden werden, ob das Wort Produkteigenschaft oder Produkteigenschaftswert ist. Aber man muss bemerken, dass ein Elternknoten der Produkteigenschaft eine Produkteigenschaft oder ein Produkt sein könnte, und ein Elternknoten des Produkteigenschaftswerts ein Produkteigenschaft, Produkteigenschaftswert oder ein Produkt sein könnte. Ein „Voting Algorithm“ könnte hier eingesetzt werden.
  Um die Produkteigenschaft und den Produkteigenschaftswert zu unterscheiden, kann man die Regel für Informationsextraktion, die in 3.2 generiert wird, kann man benutzen. Ein Voting Algorithm(mehr siehe 26) kann man benutzen, damit die Regeln bzw. die Randbedingungen entspannt werden.
- Die erstellte Hierarchie kann durch Analyse von syntaktischen Beziehungen bzw. Argumentformen verbessert werden. Zum Beispiel wenn Kindknoten und sein Elternknoten die Argumentform „conj“ haben, dann könnte die Kindknoten als Kindknoten von dem Elternknoten von seinem Elternknoten gesehen werden. Abbildung 67 illustriert die Verbesserung der Hierarchie.

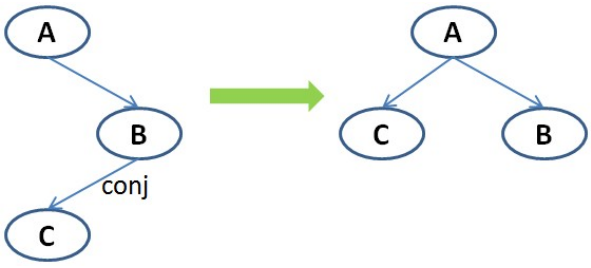


Abbildung 67 : Die Verbesserung der Hierarchie mit Argumentform „conj“.

7.3 Arbeit für die Zukunft

In dieser Diplomarbeit wird die Bestimmung der Abhängigkeit eine Abfolge der Bedingungen benutzt. Das heißt, wenn eine Bedingung verletzt, wird die Abhängigkeit falsch bestimmt. Besser kann man den Voting Algorithmus benutzen, um die Hierarchie besser aufzubauen. Es

gibt viele Wörter, die irrelevant für Produkt aber nötig für einen natürlichen Text sind. Für die zukünftige Arbeit ist es erwünscht, alle irrelevanten Wörter zu filtern. Alle in Kapitel 7.2 darstellte Probleme ist die Arbeit für die Zukunft. Dieser Diplomarbeit geht von der Analyse von „*.mln“ Datei aus, während Aktualisierung der Werte von *ClusterSum*, *CClusterSum* und *Count* wird die „*.parse“ Datei benutzt. Aber es ist vielleicht auch möglich, dass von „*.parse“ Datei ausgeht, weil „*.parse“ Datei die Bäume Struktur impliziert.

7.4 Begrenzung der System-Anforderung

Die System-Anforderung ist sehr hoch. Beim Experiment von USP mit 2000 Testdaten hat ein Rechner mit acht Cores (Intel Xeon 2.3GHz) benutzt, wenn die maximale Größe Heap von 20 GB eingestellt wird, dann braucht es 20 Minuten und 8 GB für 80 Minuten. Deshalb ist es unmöglich, USP als App s in Web oder Server einzusetzen.

8 Literaturverzeichnis

[1] C. Zietzsch und N. Zänker, Text Mining und dessen Implementierung, Diplomica Verlag GmbH, 2011, p. 12.

[2] A. Dreßler, Patente in Technologie-Orientierten Mergers & Acquisitions, 1. Auflage Hrsg., Der Deutsche Universitäts-Verlag, Juni 2006, p. 143.

[3] S. Noubours, „Computerlinguistik 13. Vorlesung," 21 01 2010. [Online]. Available: http://www.ikp.uni-bonn.de/lehre/informationen-materialien/informationen-und-materialien-kopho/materialien-1/schade/computerlinguistik/CL13%20F.pdf. [Zugriff am 01 05 2013].

[4] L. S. Zettlemoyer und C. Michael, „Learning to map sentences to logical form: Structured classification with probabilistic categorial grammers," 2005.

[5] R. J. Mooney, „Learning for Semantic Parsing," 2007.

[6] H. Poon und P. Domingos, „Unsupervised Ontology Induction from Text," 2006.

[7] H. Poon und P. Domingos, „Unsupervised Semantic Parsing," 2006.

[8] R. Ghani, K. Probst, Y. Liu, M. Krema und A. Fano, „Text Mining for Product Attribute Extraction," 2006.

[9] D. Lin, „Dependency-based Evaluation of MINIPAR," 1998.

[10] A.-M. Popescu und O. Etzioni, „Extracting Product Features and Opinions from Reviews," 2005.

[11] B. Farley, „Extracting information from free-text aircraft repair notes," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, Bd. 15, Nr. ISSN:0890-0604, pp. 295 - 305, September 2001.

[12] Z. Li und K. Ramani, „Ontology-based design information extraction and retrieval," 2007.

[13] Y. Liang, Y. Liu, C. K. Kwong und W. B. Lee, „Learning the "Whys": Discovering design rationale using text mining—An algorithm perspective," *Computer-Aided Design,* Bd. 44, Nr. j.cad.2011.08.002, p. 916–930, 10 2012.

[14] „The Stanford Parser: A statistical parser,“ [Online]. Available: http://www-nlp.stanford.edu/software/lex-parser.shtml. [Zugriff am 05 Mai 2013].

[15] P. D. N. Schweikardt, „Vorlesung Skript für Diskrete Modellierung, Kapitel 5: Logik erster Stufe,“ Wintersemester 2008/2009. [Online]. Available: http://www.tks.informatik.uni-frankfurt.de/lehre/WS0809/DM/downloads/MOD-Skript-Teil2.pdf. [Zugriff am 02 05 2013].

[16] U. Schöning, Logik für Informatiker, Spektrum Akademischer Verlag, 2000, pp. 49 - 58.

[17] M. Richardson und P. Domingos, „Markov Logic Networks,“ *Machine Learning*, Bd. 62, pp. 107-136, February 2006.

[18] H. Alshawi und J. v. Eijck, „Logical Forms In The Core Language Engine,“ 1989.

[19] G. v. Noord, „Quasi logical form,“ [Online]. Available: http://www.let.rug.nl/~vannoord/papers/nle/node20.html. [Zugriff am 02 Mai 2013].

[20] D. Jurafsky und J. Martin, „Zusammenfassung Kapitel 15: Semantik, Teil 2,“ [Online]. Available: http://www.uni-due.de/imperia/md/content/computerlinguistik/semantikkap15.pdf. [Zugriff am 02 Mai 2013].

[21] M. A. Al-Hames, „Graphische Modelle in der Mustererkennung,“ Juli 2007.

[22] R. Mikut und M. Reischl, „18. Workshop Comp Intelligence,“ pp. 5 - 7, Dezember 2008.

[23] X. Cong-Fu, H. Chun-Liang, S. Bao-Jun und L. Jun-Jie, „Research Progress in Markov Logic Networks,“ April 2011.

[24] H. Poon, „Unsupervised Semantic Parsing,“ [Online]. Available: http://research.microsoft.com/en-us/um/people/hoifung/talks/usp09.pdf. [Zugriff am 03 Mai 2013].

[25] S. Russel und P. Norvig, Künstliche Intelligenz, 3. aktualisierte Auflage Hrsg., PEARSON.

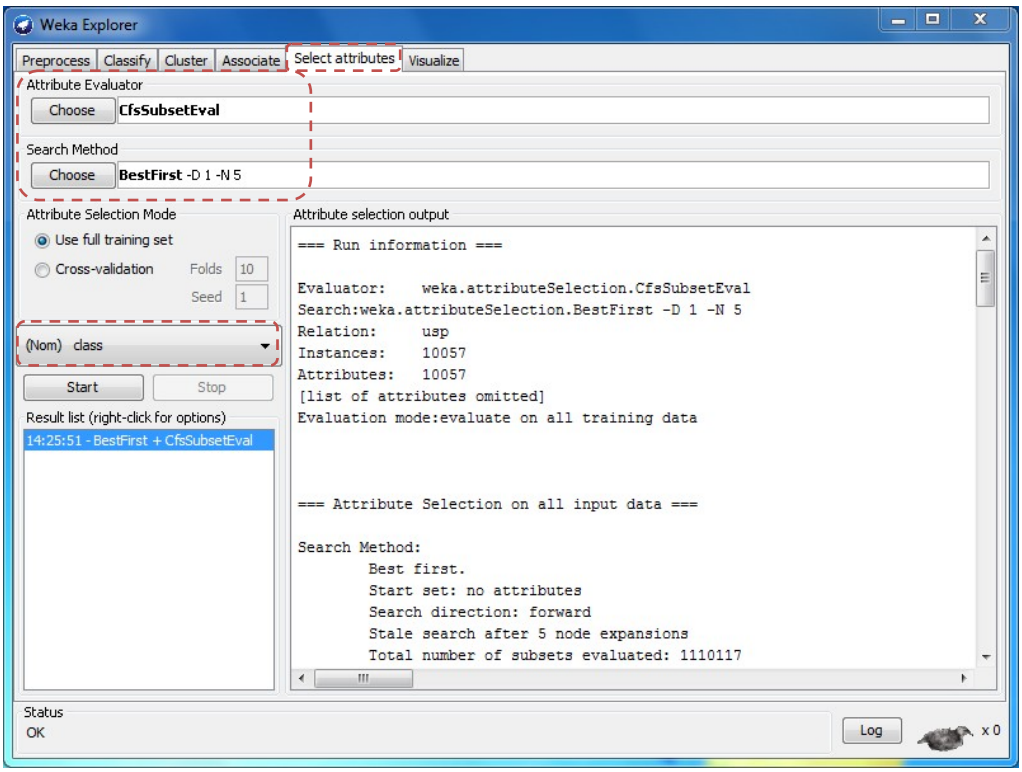
[26] W. Daelemans, „Memory-Based Language Processing. Introduction to the Special Issue,“ 1999.

[27] I. H. Witten, E. Frank und M. A. Hall, Data Mining: Practical Machine Learning Tools and Techniques, Morgan Kaufmann, 2011.

[28] I. H. Witten und E. Frank, Data Mining, Carl Hanser Verlag, 2001, p. 251.

[29] T.-L. Wong, L. Bing und W. Lam, Normalizing Web Product Attributes and Discovering, 2011.

Anhang A: Der Ergebnis von WEKA : (Aus diesem Ergebnis werden die Regeln (in 3.2) für Informationsextraktion generiert.)



Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben.
Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet.
Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens.
Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht.
Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Unterschrift:

Stuttgart, 06.05.2013