Abschlussbericht Semantic Argument Classification

Julian Baumann, Kevin Decker, Maximilian Müller-Eberstein

28. Februar 2015

1 Einleitung

Unter Semantic Argument Classification versteht man, einem Satz seine semantische Rollen wie zum Beispiel Subjekt, Prädikat und Objekt zuzuweisen. Vereinfacht gesagt versucht man zu klären, "wer wem was antut". Beispielsweise würde der Satz Ït operates stores mostly in Iowa and Nebraska"folgendermaßen annotiert werden: [$\mathbf{Arg0}^{It}$][$\mathbf{Pred}^{operates}$][$\mathbf{Arg1}^{stores}$][$\mathbf{ArgLoc}^{mostly\ in\ Iowa\ and\ Nebraska}$]. Sie gehört zum grundlegenden Pre-Processing für verschiedene Natural Language Processing-Anwendungen wie zum Beispiel Machine Translation und ist aus diesem Grunde auch ein wichtiger Aspekt der Forschung.
Angelehnt an die Veröffentlichung von [4] versuchten wir im Verlauf unseres Projekts diese Aufgabe automatisch und überwacht durch Machine Learning-Ansätze umzusetzen.

2 Grundlagen

2.1 Daten und Tools

Als Grundlage für unsere Experimente wurden verschiedene Datensätze und Tools verwendet. Wichtigste Ressourcen waren dabei PropBank und Penn TreeBank. PropBank [3] oder auch Proposition Bank ist ein Annotations-Schema, nach welchem für jedes Verb annotiert wird, welche semantische Rollen es besetzt.
Dies ist in Form von sogenannten Frames organisiert, die kodieren, welche Argumente überhaupt gefordert werden können. Im Gegensatz zu FrameNet, welches das gleiche versucht, ist PropBank jedoch nicht ganz so spezifisch sondern versucht durch algemeinere Kategorien für möglichst viele Parser verwendbar zu sein.

Die Penn TreeBank [2] ist ein syntaktisch annotierter Subkorpus des Wall Street Journals(WSJ) bestehend aus etwa einer Millionen Tokens. Davon sind

ARG0	proto-agent
ARG1	proto-patient
ARG2	instrument, benefactive, attribute
ARG3	starting point, benefactive, attribute
ARG4	ending point
ARGM	modifier

Tabelle 1: Argumente der PropBank

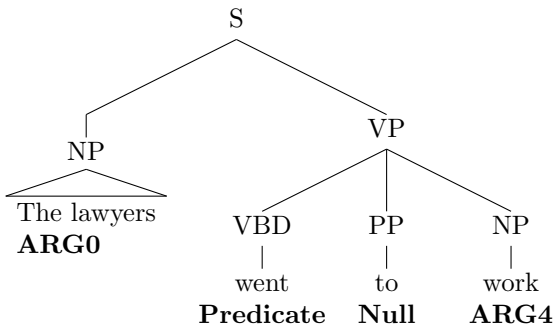
etwa 112.917 Prädikat-Argument Strukuren nach dem PropBank-Schema annotiert.
Die oben genannten Korpora bilden die Grundlage für die im Folgenden näher erklärten Experimente. Zur Extraktion der Features wurde die Programmiersprache Python in der Version 3.4 zusammen mit dem Natural Language Toolkit $Version3.0$ verwendet. Die eigentliche Anwendung der Machine-Learning Algorithmen wurde in Weka [1] realisiert.
Zur Klassifikation sollten drei verschieden Algorithmen verwendet werden: Der Naïve Bayes und der J48-Algorithmus, sowie eine Support Vector Machine. Letztere wurde ausgewählt, um den Experimentausgang mit den Ergebnissen von [4] zu vergleichen, die ebenfalls eine Support Vector Machine verwenden.

3 Umsetzung

4 Features

4.1 einzelne Features

Für das Experiment wurden 5 Features extrahiert und ihr Einfluss auf die Ergebnisse getestet. Das erste Feature, **Predicate**, ist einfach die lemmatisierte Form des Prädikats. In unserem Datenset trat dieses Feature in 3966 verschiedenen Werten auf. Zweites Feature, **Path**, beschreibt den Weg, der in einem Syntaxbaum durchlaufen werden muss, um von einem Argument zum Prädikat der Instanz zu gelangen. Dies wird durch ↑ und ↓ gekennzeichnet.

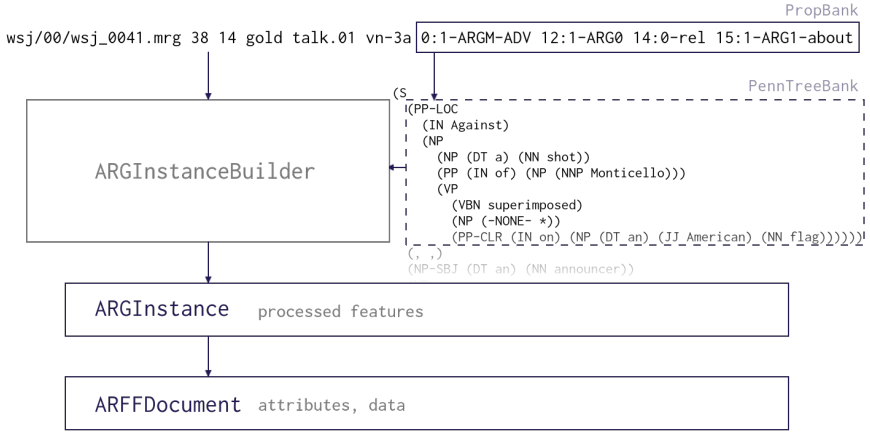


Im Satz „The lawyers went to work“ würde dieses Feature beispielsweise den Wert NP ↑ S ↓ VP ↓ VBD annehmen. Dieses Feature hat sehr viele verschiedene Ausprägungen(41737 distinct values), die durch verschiedene Satzstrukuren verursacht werden.
In der Evaluation betrachten wir ebenfalls Vor- und Nachteile der Vereinfachung dieses Features in der Form von bsp. Duplikatbeseitigung (... ↑ S ↓ VP ↓ VP ↓ ... ⇒ ... ↑ S ↓ VP ↓ ...) oder Pruning (NP ↑ S ↓ VP ↓ ... ⇒ NP ↑). Im Feature **Phrase Type** ist kodiert, welchem Konstituententypus (also NP, VP, etc.) das gesuchte Argument angehört. In unserem Fall sind dies 65 verschiedene Kategorien, nachdem für unsere Zwecke zu eingrenzende Typen konvertiert wurden (z.B. NP-SUBJ zu NP).

Die anderen beiden Features **Position** und **Voice** sollten beide intuitiv betrachtet binär sein, durch die unvollständige Annotation in Propbank ist dies im Falle von Voice, also ob das Argument passiv oder aktiv ist, nicht der Fall. Hier gibt es noch einen dritten Wert unknown, wenn für das Prädikat keine Angabe diesbezüglich gemacht wurde. Position hingegen ist ein rein binäres Feature und gibt an, ob das Argument vor oder nach dem Prädikat auftritt.

4.2 Featureextraktion

Das Ziel unserer Featureextraktion bestand darin, die Daten der PropBank und Penn TreeBank mit den oben genannten Features in normalisierter Form zusammenzuführen und in das Weka-kompatible Attribute-Relation File Format (ARFF) zu konvertieren.
Instanzen der PropBank liegen in einem Space- und Newline-separierten Textformat vor. Innerhalb einer Instanz verweisen Pointer auf Teilbäume der korrespondierenden Penn TreeBank Sätze, die in einem Klammer-Format annotiert sind.



Mit Hilfe des NLTK, welches die rohen Textdateien beider Korpora in vordefinierte Klassen lädt, erstellen wir ARGInstances, in denen alle Features in

normalisierter- und leicht abrufbarer Form vorliegen. Die Helferklasse ARGInstanceBuilder übernimmt die Zusammenführung beider Korpora, die Berechnung des Path-Features mittels Baumtraversierung und Ermittlung des Lowest-Common-Ancestors und ähnliche Extraktionsaufgaben, sowie die finale Normalisierung. Die so generierten ARGInstances werden zuletzt an ein ARFF-Document weitergegeben, welches Trainings-, Development- und Testdaten in ARFF-Dateien schreibt, die direkt mit Weka weiterverarbeitet werden können.

5 Experimente

5.1 Setup

Wir haben die Daten in 60% Training 20% Development und 20% Test Daten aufgeteilt. Als Baselines haben wir die ZeroR Baseline gewählt, da in der Literatur zumeist nur Ergebnisse für die schwierigere Aufgabe Argument Identifikation+Klassifikation zu finden sind.
Als Algorithmen benutzen wir Naive Bayes und j48 decision tree mit den Weka Default Einstellungen.
Ursprünglich hatten wir auch noch geplant auf einem SVM basierten Algorithmus zu evaluieren. Dabei wurden jedoch alle Instanzen der häufigsten Klasse zugeteilt, was an einem unbalancierten Datenset liegen könnte.

5.2 Feature Selektion auf dem Development set (Naïve Bayes

	Precision	Recall	F-Measure	F-Measure Change
All Features	*0.771*	*0.778*	*0.770*	*0*
-voice	0.748	0.754	0.745	-0.025
-path	0.778	0.783	0.776	**+0.006**
-phraseType	0.735	0.747	0.733	-0.037
-position	0.758	0.773	0.757	-0.013
-predicate	0.717	0.732	0.716	**-0.054**

Auf dem Development Set konnten wir feststellen, dass wir durch das Entfernen des Path Features bessere Ergebnisse erreichen konnten. Auch weitere Versuche das Path Feature zu verbessern, indem wir aufeinanderfolgende, doppelte Kategorien zusammenfassen und nur den Pfad vom Argument aufwärts benutzen, schlugen fehl. Effektiv verschlechterte das Feature sogar unsere Ergebnisse mit einem Abfall auf 0.738 für den Naïve Bayes und 0.838 für den J48-Algorithmus im F-Measure
Deshalb ließen wir dieses Feature, obwohl es im Originalpaper mit einbezogen wurde, aus unserer Evaluierung auf dem Testset raus, um ein besseres Ergebnis zu erzielen.

5.3 Test Set Ergebnisse

	Precision	Recall	F-Measure
Baseline	*0.132*	*0.364*	*0.194*
Naïve Bayes	0.777	0.781	0.774
j48 Tree	0.847	0.849	0.846

Die von uns mit Bedacht niedrig gelegte ZeroR Baseline konnten wir, wie zu erwarten war, definitiv schlagen. Von den beiden Algorithmen, die wir nach mehreren Durchläufen für am passendsten erachtet hatten, erwies sich der Decision Tree Algorithmus als deutlich effektiver als der Naïve Bayes Algorithmus. Hierbei wird ARG0 sehr präzise klassifiziert und weitet sich dabei allerdings auch merklich in die Instanzen der ARG1-Klasse aus, was bei der letzteren zu einem leichten Abfall im Recall führt. Bei der Unterscheidung von Agens und Patiens besteht also ein geringfügiger Bias zu einer Klassifizierung als Agens. Alle weiteren Klassen zeigen stets die größten Überschneidungen mit ihren semantisch ähnlichsten Nachbarn (ARG3 und ARG2, ARG5 und ARGM etc.), was also bedeutet, dass selbst bei einer Missklassifikation die etwaige Valenz beibehalten wird.
Insgesamt zeigt sich also, dass, wie zu erwarten, die häufigsten Argumentklassen am besten trainiert und folglich auch am präzisesten klassifiziert werden und sich die semantische Ähnlichkeit, insbesondere bei den vom Prädikat weiter entfernten Klassen, ebenfalls stark in der Ähnlichkeit der Feature-Werte widerspiegelt.

6 Retrospektive

Literatur

[1] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[2] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *COMPUTATIONAL LINGUISTICS*, 19(2):313–330, 1993.

[3] M. Palmer, D. Gildea, and P. Kingsbury. The proposition bank: An annotated corpus of semantic roles. *Comput. Linguist.*, 31(1):71–106, Mar. 2005.

[4] S. Pradhan, K. Hacioglu, V. Krugler, W. Ward, J. H. Martin, and D. Jurafsky. Support vector learning for semantic argument classification, 2005.