AaS - Annotation als Suchproblem

Rebekka Hubert Michael Staniek Simon Will

December 6, 2016

1 Introduction

In diesem Dokument werden die Funktion, die Funktionsweise und die Systemvoraussetzungen der Software „Annotation als Suche“ (AaS) beschrieben.

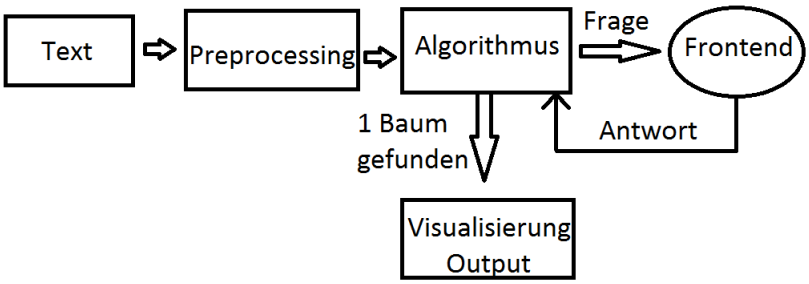
1.1 Motivation

Bei AaS handelt es sich um eine Software zur Unterstützung von Annotatoren, die auf Basis eines Parseforest Fragen an den Annotator stellt und anhand dieser den optimalen Parsebaum auswählt. Dafür benötigt der Annotator zwingend Grunderfahrung im Annotieren, aber keinerlei Programmierkenntnisse. Dies beschleunigt den Vorgang der menschlicher Dependenzannotation, die weiterhin die beste Qualität aufweist.

1.2 Architecture

2 System

Um die größtmögliche Kompatibilität und Flexibilität zu erreichen, besteht das System aus formal drei Teilbereichen, die durch Schnittstellen kommunizieren. Dies ermöglicht es, Konkret handelt es sich um eine Server-Client-Konstruktion, die es dem User ermöglicht, sowohl das gesamte System als auch lediglich Teilbereiche zu installieren: So ist das Preprocessing optional und auch der Client kann für einen bereits installierten Server aufgesetzt werden. Im Folgenden ein Überblick über das gesamte System:



Das System besteht bekanntlich aus drei Teilbereichen, die durch Schnittstellen kommunizieren:

- Preprocessing
  Generieren der möglichen Parsebäume und Übergabe der k-besten an die Fragengenerierung über die Schnittstelle Preprocessing - Fragegenerierung
- Algorithmus zur Fragengenerierung auf einem Server
  erhält die k-besten Parsebäume
- Client
  Übermittlung der Fragen an den Annotator und der Antwort des Annotators an den Algorithmus über die Schnittstelle Client - Server sowie vom Prozess unabhängige Darstellung des Parsebaums

2.1 AaS-Server

Der AaS-Server generiet die Fragen, nach welchen der optimale Parsebaum ausgeählt wird. Der Algorithmus folgt dabei folgendem Schema:

- Generiere aus Parseforest die Frage, welche bei ihrer Beantwortung die meisten Bäume rausfiltert.
  - Suche des Tupels, das den Suchraum am ehesten halbiert:

    a $\leftarrow$ len(parses) Anzahl der Parse-Bäume.
    For each tuple
      score $\leftarrow$ abs(count(tuple) $-\ \frac{a}{2}$)
    End For

- Nimm den Tupel als Frage

Näheres im AaSP.

2.1.1 Dependencies

Der Algorithmus benötigt folgende Software um korrekt zu laufen:

- Python $\geq$ 3.4
- Python3 packages: asyncio, Flask
- Parser, der k-best Parses liefert (Anders Bjorkelunds transition-based parser)
- TCP- oder UNIX-Sockets
- JSON

2.1.2 Algorithm

2.2 AaS-CLI-Client

2.2.1 Dependencies

2.3 AaS-Web-Client

Anmerkung:Der AaS-Web-Client befindet sich zu diesem Zeitpunkt noch im frühen Entwicklungsstadium, folglich besteht keine Garantie, dass alle hier getroffen Aussagen fristgerecht umgesetzt werden können. Dementsprechend ist der folgende Abschnitt unterteilt: Zuerst werden die Features des Client aufgeführt, die in der finalen Version zur Verfügung stehen werden. Danach folgen jene Features, die je nach vorhandenen zeitlichen Rahmen noch folgen werden.
Der AaS-Web-Client ist eine interaktiven GUI und handhabt die Kommunikation Benutzer - Software. Über den Client werden sowohl die Benutzereingaben entgegengenommen und an den AaS-CLI-Client weitergeleitet als auch dessen Systemausgaben dem Benutzer zugeführt. Dabei werden dem Benutzer durch den Client mehrere Funktionen zur Verfügung gestellt, die die Annotationsaufgabe erleichtern:
Zuerst einmal werden über den Client die Fragen des Algorithmus in menschenlesbarer Form ausgegeben und können vom Benutzer mit „j “ oder „nein“ beantwortet werden. Sobald der optimale Parsebaum ausgewählt wurde, wird dieser über den Web-Client ausgegeben. Dieser hat dann die Möglichkeit diesen abzuspeichern oder komplett zu verwerfen.
Alle weiteren Features fallen in den optionalen Bereich.
Nachdem der optimale Baum ausgewählt wurde, erhält der Benutzer die Option diesen abzuändern. Des Weiteren sind noch weitere Benutzeraktionen geplant, *undo* und *abort*. Wird *undo* ausgeführt, wird die letzte Benutzeraktion rückgängig gemacht. Eine bereits beantwortete Frage kann somit ein weiteres Mal beantwortet und ein etwaiger Fehler behoben werden. *abort* beendet den Hilfsvorgang, wobei die übrigen Bäume beibehalten werden. Der Benutzer hat dann die Möglichkeit sich entweder den zu diesem Zeitpunkt intern als optimal gekennzeichneten Baum anzeigen zu lassen oder sich die durch die zuvor beantworteten Fragen festgelegten Kanten anzusehen.

2.3.1 Dependencies

Der Web-Client hat bis auf weiteres folgende Dependency:

- python-package Flask

3 Evaluation

Die Evaluation erfolgt automatisch, da dadurch sowohl in der gleichen Zeitspanne mehr evaluiert werden kann als es von Hand möglich wäre als auch menschliche Fehler vermieden werden.
Der Goldparse eines Satzes wird hierbei als Ersatz für einen menschlichen Annotator benutzt. Jegliche Fragen des Servers werden an diesen Goldparse gestellt. Sobald der Algorithmus den optimalen Parse gewählt hat, wird dieser mit dem Goldparse verglichen.

3.1 Evaluationsmaße

Die Evaluation erfolgt über die Anzahl der korrekt gelabelten Kanten. Zur Bewertung der Ergebnisse werden insgesamt drei EvaluationsmaSSe benutzt:

- Minimum Edit Distance
  Anzahl der nicht übereinstimmenden, gelabelten Kanten
- Labeled Attachment Score
  Anzahl der gelabelten Kanten, die mit dem Goldstandard übereinstimmen
- Error Counter
  Anzahl der nicht gefundenen Bäume (die Antworten auf die Fragen haben jede möglich Baum ausgeschlossen)