Open Information Extraction using Wikipedia

Soraya Nikousokhan

Department of Computer Science
Albert-Ludwigs-University Freiburg, Germany

Nov. 2013
Overview

• Motivation
• Wikipedia-based Open Extractor
• Architecture of WOE
  • Preprocessor
  • Matcher
  • Learning Extractors
    • Extraction with parser features
    • Extraction with Shallow Features
• Performance Analysis
• Shallow or Deep Parsing
• Conclusion
Motivation

- **IE systems** extract semantic relations from natural language text
- Use **supervised learning**
  - availability of training data
  - Can not scale to the web
- Open IE systems aim to handle **the unbounded number of relations**
  - self-supervised learning
  - Automatic heuristics generate labeled data
- **How well** can these open systems perform?
Wikipedia-based Open Extractor

• Improves dramatically on text runner’s *precision* and *recall*.

• **A self-supervised learning**
  • heuristic matches between Wikipedia infobox attribute values and corresponding sentences

• **Operate in two modes:**
  • Restricted to POS
  • Dependency parse features
Open Information Extractor

- A function from a document $d$, to a set of triples, $\{<\text{arg1}, \text{rel}, \text{arg2}>\}$, where the **args** are **noun phrases** and “**rel**” indicating a semantic **relation** between the two noun phrases.

- The extractor should produce **one triple** for **every relation** stated explicitly in the text.
Open Information Extractor Example

• Article: “Stanford university”
• Infobox: <established,1891>
• Sentence: ”the university was founded in 1891 by…”
• The triple would be:
  • <arg1,rel,arg2>
  • <Stanford university,established,1891>
Architecture of WOE
Preprocessor

- **Sentence Splitting**
  - Transform each Wikipedia article into HTML
  - Splits into sentences by OpenNLP

- **NLP annotation**
  - OpenNLP to supply POS tags and NP-chunk annotations
  - Stanford Parser to create a dependency parse

- **Compiling synonyms**
  - The preprocessor build a set of synonyms
  - Uses Wikipedia redirection pages and backward links
Matcher

• Constructs **training data** for the learner component

• Given a **Wikipedia page** with an **infobox**
  • the matcher **iterates** through all **infobox attributes**
  • looking for sentence that contains **references to both the subject of the article** and the **attribute value**
  • These noun phrases will be annotated in the training set.
Matcher

- Matching primary entities
  - Use heuristics:
    - Full match
    - Partial match: “Amherst ” matches “Amherst, Mass”
    - Patterns of “the<type>”: “City” for ” Ithaca”
    - The most frequent pronoun: ” he” for the page on “Albert Einstein”

- Matching sentence
  - The matcher seeks a unique sentence to match the attribute value
Learning Extractors

- **Extraction with parser features**
  - WOE$_{Parse}$ using features from dependency-parse trees.
  - It uses a pattern learner to classify whether the shortest dependency path between two noun phrase indicate a semantic relation

- **Extraction with shallow features**
  - WOE$_{POS}$ limited to shallow features like POS tags
  - Trains a conditional random field (CRF) to output certain text between noun phrases when the text denotes such a relation.
Extraction with Parser Features
Shortest Dependency Path as Relation

“Dan was not born in Berkeley.”

- The Stanford parser dependencies are:

  - nsubjpass(born-4, Dan-1)
  - auxpass(born-4, was-2)
  - neg(born-4, not-3)
  - root(ROOT-0, born-4)
  - prep_in(born-4, Berkeley-6)
Extraction with Parser Features
Shortest Dependency Path as Relation

• “Dan was not born in Berkeley.”

• **CorePath:**

```
Dan  nsubjpass  born  prep_in  Berkeley
```

• **ExpandPath:**

```
Dan  nsubjpass  born  prep_in  Berkeley
   was  auxpass
   neg   not
```
Extraction with Parser Features
Building a Database of Patterns

- Learner generates a **corePath** between the tokens denoting the **subject** and the **infobox attribute value**.

- To improve **learning performance**:
  - **Generalized–corePaths**: eliminate irrelevant relations
  - Lexical words in corePaths are replaced with their POS tags
  - Extraction pattern “$N \xrightarrow{nsubjpass} V \xleftarrow{prep} N$”

- WOE builds a **database** (named $DB_p$) of 15,333 distinct patterns

- Each **pattern $p$** associated with a **frequency**
Extraction with Parser Features
Learning a Pattern Classifier

- $WOE^{parse}$ checks whether the generalized-corePath from a test triple is present in $DB_p$ and computes the normalized logarithmic frequency as the probability:
  - $f_p$: associated frequency of the pattern
  - $f_{max}$: maximal frequency of patterns in $DB_p$
  - $f_{min}$: controlling threshold, minimal frequency of a valid pattern

$$w(p) = \frac{\max(\log(f_p) - \log(f_{min}), 0)}{\log(f_{max}) - \log(f_{min})}$$
Extraction with Parser Features
Learning a Pattern Classifier

• Example:
  • “Dan was not born in Berkeley ”
  • Dan as arg1, Berkeley as arg2
  • Computes corePath \[ \text{Dan} \overset{\text{nsbjpass}}{\to} \text{born} \overset{\text{prep in}}{\leftrightarrow} \text{Berkeley} \]
  • Abstracts to “\( N \overset{\text{nsbjpass}}{\to} V \overset{\text{prep}}{\leftrightarrow} N \)”
  • Queries \( DB_p \) to retrieve the frequency \( f_p = 29112 \) and assigns probability of 0.95\( (f_m : 50.259) \)
  • \( WOEparse \) traverses the triple’s expandPath to output the final expression \( \langle \text{Dan, wasNotBornIn, Berkeley} \rangle \)
Extraction with Shallow Features

• **High speed** can be **crucial** when processing web-scale corpora
• Shallow features like **POS-tags**
• Use the same matching sentence set behind $DB_P$ to generate **positive examples**
• **Negative examples** are generated from **random** noun-phrase pairs
  • generalized-corePaths which are **not in** $DB_P$
• **Learning algorithm** and selection features as textrunner
  • A two-order CRF chain model is trained with Mallet package
Experiments

- **Three corpora** for experiments:
  - WSJ from Penn Treebank
  - Wikipedia
  - Web
- Randomly selected **300 sentences** for each
- Examined by two people to **label** all **reasonable triples**
- Submitted to **Amazon Mechanical Turk** for **verification**
- Each triple examined by **5 Turkers**
- **Positive** when more than **3 Turkers** marked them as positive
Overall Performance Analysis

- How do these systems *perform* against each other?

- How does *performance* vary w.r.t *sentence length*?

- How does *extraction speed* vary w.r.t *sentence length*?
Overall Performance Comparison

- $\text{WOE}^{\text{pos}}$ is better than TextRunner on precision
  - Better training dataset

- $\text{WOE}^{\text{parse}}$ is the best on recall
  - Parser features
Extraction Performance vs. Sentence Length

- Long sentence have long-distance relations
- Difficult for shallow feature
Extraction Speed vs. Sentence Length

- $WOE^{parse}$'s extraction time grows quadratically
- Reliance on parsing
Shallow or Deep Parsing

- Shallow features like **POS tags** enable **fast extraction** over large-scale corpora
- Deep features are derived from **parse trees**
  - **training better** extractors
- Abstracted dependency path features are **highly informative**
- In Web, many sentences contain **complicated long-distance relations**
  - **Parser features** are more **powerful**
Conclusion

• **WOE** a new approach that uses **self-supervised learning** over **unlexicalized features**, based on **heuristic match** between Wikipedia infoboxes and corresponding text

• Runs in **two** modes
  • **WOE\textsuperscript{pos}**: a CRF extractor trained with shallow features like POS tags
  • **WOE\textsuperscript{parse}**: a pattern classifier learned from dependency path patterns

• In comparison with **textrunner**
  • **WOE\textsuperscript{pos}** runs at **the same speed**, but achieves an **F-measure** which is between 9% and 23% **greater**
  • **WOE\textsuperscript{parse}** achieves an **F-measure** which is between %51 and 70%, but runs about 30X times **slower** due to its reliance on parsing.
Reference