




Unsupervised Relation Extraction from Web

-Bhavishya Mittal (I I I 98)

- Vempati Anurag Sai (Y9227645)

- 
- Problem Statement
 - Previous Work
 - Approach
 - Self learning
 - Extractor
 - Probability
 - Query
 - Work Done
 - Work Remaining
 - Dataset

Problem Statement

Extracting relation tuples from an unstructured corpus that is effective at noise removal.

During the query process, given a partially filled tuple, our system will search for possible entries for the missing fields and rank the resulting tuples based on a probabilistic measure.

Previous Work

- Previously decided set of relations.
- Supervised vs unsupervised.
 - Supervised: Manual annotations(tiresome)
/wikipedia infobox(domain specific)
- Heavy linguistic machinery. Don't scale properly to web data.

Approach

- Work is divided into 3 steps :
 - Self-Supervised Learner
 - Given a small corpus sample as input, the Learner outputs a classifier that labels candidate extractions as “trustworthy” or not. The Learner requires no hand-tagged data.
 - Single-Pass Extractor
 - The Extractor makes a single pass over the entire corpus to extract tuples for all possible relations. The Extractor does not utilize a parser. The Extractor generates one or more candidate tuples from each sentence, sends each candidate to the classifier, and retains the ones labeled as trustworthy.
 - Redundancy-Based Assessor
 - Group similar tuples to get a frequency count. Then, assign a probability to each retained tuple.

Approach: Self-Supervised Learner

- Two Broad steps:
 - Automatically labeling its own training data as positive or negative.
 - Using this labeled data to train a classifier, which is then used by the Extractor module.

Deploying a deep linguistic parser to extract relationships between objects is not practical at Web scale. The classifier is also efficient at parser's noise removal. So, the parser is used to train the classifier.

Self-Supervised Learner : Step I

- Extractions take the following form

$$\text{tuple } 't' = (e_i, r_{i,j}, e_j)$$

Where e_i and e_j are string meant to denote entities, and $r_{i,j}$ is a string meant to denote a relationship between them.

- Some of the heuristics used to identify any tuple as trustworthy or not are:
 - The length of the dependency chain between e_i , e_j and $r_{i,j}$.
 - Neither e_i nor e_j consist solely of a pronoun.

Self-Supervised Learner : Step II

- In this step our task is to train a SVM classifier from the training data we obtained by labeling some set of relations as trustworthy or not.
- Set of tuples of the format $= (e_i, r_{i,j}, e_j)$, are mapped to a feature vector representation.
- Some features used are:
 - The presence of part-of-speech tag sequences in the relation $r_{i,j}$
 - The number of tokens in $r_{i,j}$
 - The number of stopwords in $r_{i,j}$
 - Whether or not an object is found to be a proper noun
 - The POS tag to the left of e_i , or the POS to the right of e_j

Approach: Single-Pass Extractor

- The Extractor makes a single pass over its corpus, automatically tagging each word in each sentence with its most probable part-of-speech.
- Using these tags, entities are found by identifying noun phrases.
- Relations are found by examining the text between the noun phrases and heuristically eliminating non-essential phrases such as adjective or adverb phrases.
- Finally, each candidate tuple 't' is presented to the classifier. If the classifier labels it as trustworthy, it is extracted and stored.

Approach: Redundancy-Based Assessor

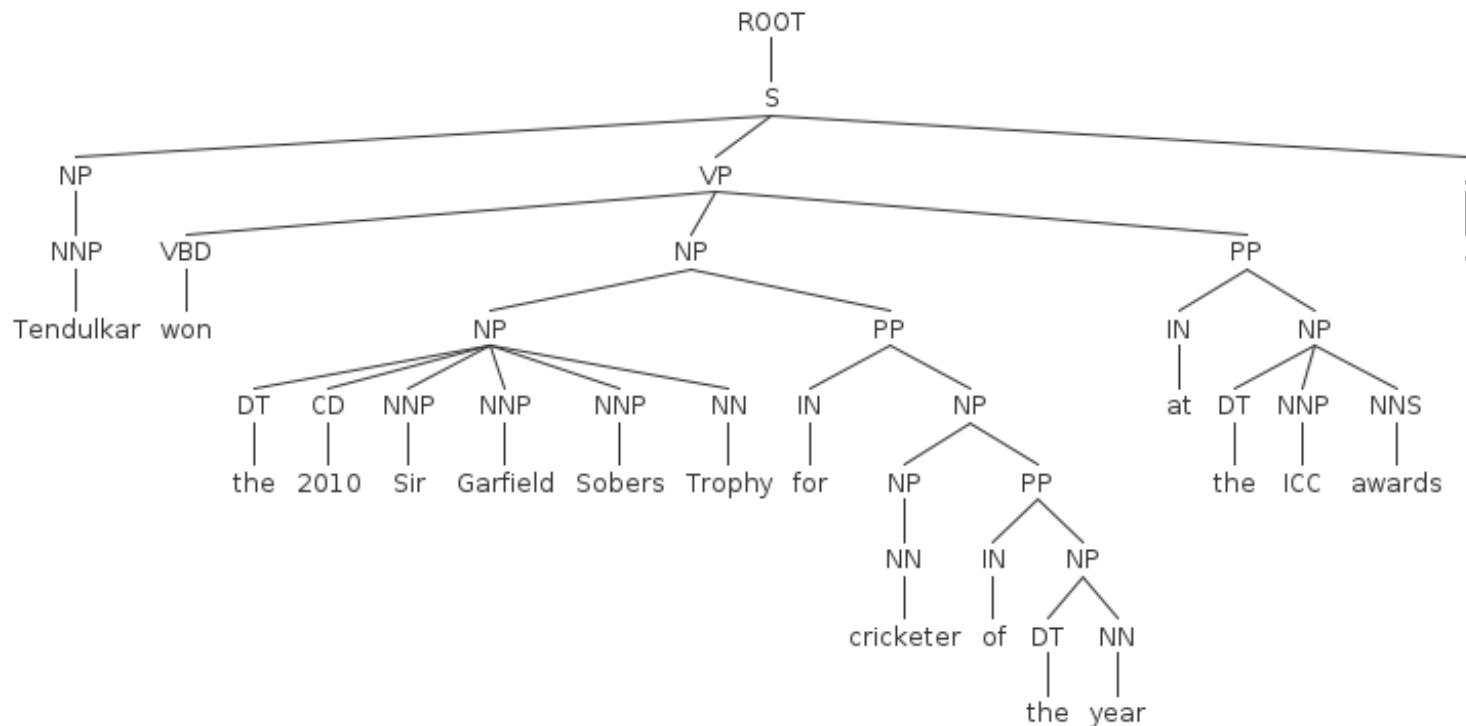
- Run through all the tuples obtained by the extractor module and merge similar ones.
- Estimate the probability that a tuple $t = (e_i, r_{i,j}, e_j)$ is a correct instance of the relation $r_{i,j}$ between e_i and e_j given that it was extracted from k different sentences.

Work Done

- Run Stanford POS Tagger on set of sentences picked randomly from wikipedia.
 - We get tags for each word and dependency tree for the sentence.
- Using these words and dependency graph we picked entities to be used as e_i and e_j and the relation ie $r_{i,j}$ between them.
 - Used dijkstra's algorithm for computing the minimum distance between two entries in the dependency graph.
 - In this algorithm we used the weight on the edges depending on the relation given by Stanford Dependency Parser.
- Training of the SVM classifier

Work Done : Continued

- Input sentence: “Tendulkar won the 2010 Sir Garfield Sobers Trophy for cricketer of the year at the ICC awards.”



Work Done : Continued

- Input sentence: “Tendulkar won the 2010 Sir Garfield Sobers Trophy for cricketer of the year at the ICC awards.”

Collapsed dependencies:

```
Head: won (2); dependent: Tendulkar (1); relation: nsubj
Head: ROOT (0); dependent: won (2); relation: root
Head: Trophy (8); dependent: the (3); relation: det
Head: Trophy (8); dependent: 2010 (4); relation: num
Head: Trophy (8); dependent: Sir (5); relation: nn
Head: Trophy (8); dependent: Garfield (6); relation: nn
Head: Trophy (8); dependent: Sobers (7); relation: nn
Head: won (2); dependent: Trophy (8); relation: dobj
Head: Trophy (8); dependent: cricketer (10); relation: prep_for
Head: year (13); dependent: the (12); relation: det
Head: cricketer (10); dependent: year (13); relation: prep_of
Head: awards (17); dependent: the (15); relation: det
Head: awards (17); dependent: ICC (16); relation: nn
Head: won (2); dependent: awards (17); relation: prep_at
```

Work Done : Continued

- When we used only single-word noun for ei and ej , we obtained unsatisfactory results as shown below:

```
[(Tendulkar, 'won', Trophy), (Tendulkar, 'won at', awards), (Trophy, 'won at', awards)]
```

```
[(Tendulkar, 'won', Sir), (Tendulkar, 'won', Garfield), (Tendulkar, 'won', Sobers), (Tendulkar, 'won', cricketer), (Tendulkar, 'won', year), (Tendulkar, 'won', ICC), (Sir, 'won', Garfield), (Sir, 'won', Sobers), (Sir, 'won', Trophy), (Sir, 'won', cricketer), (Sir, 'won', year), (Sir, 'won', ICC), (Sir, 'won at', awards), (Garfield, 'won', Sobers), (Garfield, 'won', Trophy), (Garfield, 'won', cricketer), (Garfield, 'won', year), (Garfield, 'won', ICC), (Garfield, 'won at', awards), (Sobers, 'won', Trophy), (Sobers, 'won', cricketer), (Sobers, 'won', year), (Sobers, 'won', ICC), (Sobers, 'won at', awards), (Trophy, 'won', cricketer), (Trophy, 'won', year), (Trophy, 'won', ICC), (cricketer, 'won', year), (cricketer, 'won', ICC), (cricketer, 'won at', awards), (year, 'won', ICC), (year, 'won at', awards), (ICC, 'won at', awards)]
```

Work Done : Continued

- To rectify this problem we used NP Chunking i.e whole Noun Phrase as our e_i and e_j .

('Tendulkar', 'won', 'the 2010 Sir Garfield Sobers Trophy for cricketer of the year')

('Tendulkar', 'won at', 'the ICC awards')

('the 2010 Sir Garfield Sobers Trophy for cricketer of the year', 'won at', 'the ICC awards')

Work Remaining

- Verifying the classifier
- Running Single-Pass Extractor
- Applying probabilities to each tuple
- Evaluation

Dataset

- Wikipedia

References

- Banko, Michele, et al. “Open Information Extraction from the Web.” IJCAI. Vol. 7. 2007.
- Fader, Anthony, Stephen Soderland, and Oren Etzioni. “Identifying relations for open information extraction.” Proceedings of the Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, 2011.
- Dan Klein and Christopher D. Manning. 2003. Accurate Unlexicalized Parsing. Proceedings of the 41st Meeting of the Association for Computational Linguistics, pp. 423-430.
- Marie-Catherine de Marneffe, Bill MacCartney and Christopher D. Manning. 2006. Generating Typed Dependency Parses from Phrase Structure Parses. In LREC 2006.
- Jython libraries for Stanford Parser by Viktor Pekar
- Python implementation of Dijkstra’s algorithm by David Eppstein UC Irvine, 4 April 2002