Parser Extraction of Triples in Unstructured Text

Shaun D'Souza
Technical Lead, Wipro Limited, Bangalore, Karnataka, India

Article Info

Article history:

Received Aug 14, 2016
Revised Oct 18, 2016
Accepted Nov 21, 2016

Keyword:

NLP
Open information extraction
Relation extraction

ABSTRACT

The web contains vast repositories of unstructured text. We investigate the opportunity for building a knowledge graph from these text sources. We generate a set of triples which can be used in knowledge gathering and integration. We define the architecture of a language compiler for processing subject-predicate-object triples using the OpenNLP parser. We implement a depth-first search traversal on the POS tagged syntactic tree appending predicate and object information. A parser enables higher precision and higher recall extractions of syntactic relationships across conjunction boundaries. We are able to extract 2-2.5 times the correct extractions of ReVerb. The extractions are used in a variety of semantic web applications and question answering. We verify extraction of 50,000 triples on the ClueWeb dataset.

Corresponding Author:

Shaun D'Souza,
Technical Lead (CTO Office)Wipro Limited,
Bangalore, Karnataka, India
Email: shaun.dsouza1@wipro.com

1. INTRODUCTION

There is a considerable amount of research in natural language processing (NLP). With the availability of a larger set of NLP tools like OpenNLP [3], it is today possible to POS tag and chunk vast amount of unstructured text that is available on the internet. Projects like ClueWeb, OpenIE and Wikipedia provide a corpus of text data which can be used for ontological engineering. OpenNLP supports the POS tagging and chunking of data. It outputs a parse tree for the data which encapsulates the syntactic content in a n-ary tree data structure. POS tag data provides a higher level of understanding as compared to a bag of words approach to web search today. We explore opportunities for language inference and understanding through subject-predicate-object analysis of web scale unstructured data.

Various methods are used to extract subject-predicate-object triples in unstructured data. DBpedia extractor is used to generate triples using annotated field information in Wikipedia. OpenIE [1] used POS and chunker data while ClauseIE [2] uses a parser to output a set of word triples.

Bootstrapping functions use N-gram models to generate a template for a given combination of noun phrases. These are used to search a larger corpus of data for similar templates and generate values. NER taggers are used to annotate person and location information.

We assume a context free grammar (CFG) for English language [4].

$G = (N, \Sigma, R, S)$

$N \in$ {non-terminal symbols}
$\Sigma \in$ {terminal symbols}
$R \in$ {rules} of the form $X \rightarrow Y_1Y_n$ for $n \geq 0$, $X \in N$, $Y_i \in (N \cup \Sigma)$
$S \in N$ start symbol {TOP}
$N =$ {S, NP, VP, PP, DT, VB, NN, IN}

S = S
Σ = word in the English language

R = S → NP VP
VP → VB
VP → VB NP
VP → VP PP
NP → DT NN
NP → NP PP
PP → IN NP

2. RESEARCH METHOD

We found a limitation of extractors that were unable to extract the verb phrase accurately and instead appended a large amount of additional words including the trailing noun and preposition context. The extractors were unable to process sentence and conjunction values resulting in incorrect verb and object phrases. A parse tree is able to capture conjunction and object phrase information correctly. Although there is an overhead on the parsing time.

We evaluate the parser tree for sequences of NP noun phrases (subject, object) and VB - verbs (predicate). OpenNLP generates a parse tree using the CFG rules. We implement an in-order traversal of the syntactic tree to detect SVO phrases. We maintain a list of all NP phrases in the sentence. We then traverse the tree to detect subject object pairs and the predicate.

```
function SUBJECT-NOUN-PHRASE(parse)
    kids ← CHILD(parse)
    for i = 1 to SIZE(kids) do
        if TYPE(kids[i]) = NP then
            subject = kids[i]
            for j = i + 1 to SIZE(kids) do
                if TYPE(kids[j]) = VP | PP | SBAR then
                    explored ← an empty set
                    while kids[j] not in explored do
                        extraction ← APPEND(subject, PREDICATE-VERB-PHRASE(kids[j]))
                        PRINT(extraction)

        SUBJECT-NOUN_PHRASE(kids[i])

function PREDICATE-VERB-PHRASE(parse) returns solution, failure
    kids ← CHILD(parse)
    initialize predicate string to be empty

    for i = 1 to SIZE(kids) do
        if TYPE(kids[i]) = VP | S then
            if kids[i] not in explored then
                return APPEND(predicate, PREDICATE-VERB-PHRASE(kids[i]))
        else if TYPE(kids[i]) = VB | JJ | RB | MD | TO | ADVP | DT | NN | IN then
            predicate ← APPEND(predicate, kids[i]);

            for j = i + 1 to SIZE(kids) do
                if TYPE(kids[j]) = NP | PP | ADJP | S | SBAR then
                    return APPEND(predicate, OBJECT-NOUN_PHRASE(kids[j]))
    add parse to explored
    return failure
```

Figure 1. Subject-Predicate Phrase Algorithm

```
function OBJECT-NOUN-PHRASE(parse) returns solution, failure
    found ← false
    kids ← CHILD(parse)
    initialize object string to be empty

    for i = 1 to SIZE(kids) do
        if TYPE(kids[i]) = NP | S then
            found ← true
            if kids[i] not in explored then
                return APPEND(object, OBJECT-NOUN-PHRASE(kids[i]))
            else
                return APPEND(object, GET-COVERED-TEXT(kids[i]))
        else if TYPE(kids[i]) = PP then
            if kids[i] not in explored then
                return APPEND(object, OBJECT-PREPOSTION-PHRASE(kids[i]))
            else
                return APPEND(object, GET-COVERED-TEXT(kids[i]))
        else if TYPE(kids[i]) = IN | TO then
            object ← APPEND(object, kids[i])

    add parse to explored
    if not found and TYPE(parse) = NP then
        return APPEND(object, parse)

    return failure

function OBJECT-PREPOSITION-PHRASE(parse) returns solution, failure
    kids ← CHILD(parse)
    initialize preposition string to be empty

    for i = 1 to SIZE(kids) do
        if TYPE(kids[i]) = NP and not in explored then
            return APPEND(preposition, OBJECT-NOUN-PHRASE(kids[i]))
        else if TYPE(kids[i]) = PP and not in explored then
            return APPEND(preposition, OBJECT-PREPOSTION-PHRASE(kids[i]))
        else if TYPE(kids[i]) = IN | TO | JJ | ADVP then
            preposition ← APPEND(preposition, kids[i])

    add parse to explored
    return failure
```

Figure 2. Object Phrase Algorithm

We implement a depth-first search on the n-ary parse tree. We search the parse tree for a noun-verb phrase indicating the subject-predicate -

Figure 1. The noun phrase is used as the subject in the clause. We look for a verb phrase VP or preposition phrase PP in the siblings. In the case of subsequent conjunctions CC and WHNP phrases, we continue to search the sibling nodes. For all found VP, PP we search for the predicate clause in the sentence. A predicate clause consists of a sequence of verb, adjectives, adverb and modal identifiers. These are appended to a string of predicates. VP phrases are searched recursively till we find a terminal NP object clause. We represent the SVO in the triples format. We use a training set of 200 phrases from earlier publications on information extraction. These give us a range of parse trees to evaluate the search on and refine.

Earlier work on information extraction was limited to the capabilities of the POS and Chunker tags. Verb phrases were detected using statistical probabilities of frequently occurring patterns in the English language. We implement a rigorous parse tree design which preserves the language syntax of the text data.

As there is a high availability of computing today in the cloud, we implement the SVO parser as an offline function to process the syntactic tree. We parse all the sentences in the text and generate a parsed output. This is subsequently used to generate the SVO triples. With the availability of computing we can improve performance of the parser by parallelizing the parsing of input sentences.

We contrast the SVO triples with past research including OpenIE and ClauseIE. We find that a parser based approach is able to extract a large number of SVO's accurately. Availability of a syntactic parse tree also enables us to extract triples with reduced ambiguity. The obtained triples map exactly to sub-trees in the sentence parse tree and capture all the semantic information – subject predicate. The n-ary parse tree encapsulates the syntactic structure of the sentence completely.

We are able to precisely extract SVO information. In the initial revision of the code we implemented predicate extractions to include the trailing noun phrase. This was updated to resolve the object clause to contain the noun phrase NP and a trailing preposition phrase PP - Figure 2. We use a set of heuristics to maximize the number of triples generated for each noun phrase, verb phrase.

3. RESULTS AND ANALYSIS

The SVO extractions are coherent as OpenNLP captures the language syntax in the parse tree. We compare the number of extractions with the ReVerb extractor. We observe a larger number of triples as we are searching for all noun phrases in the object. The NLP parser is able to extract a large number of triples matching ReVerb and ClausIE.

Example sentence

The principal opposition parties boycotted the polls after accusations of vote rigging, and the only other name on the ballot was a little known challenger from a marginal political party

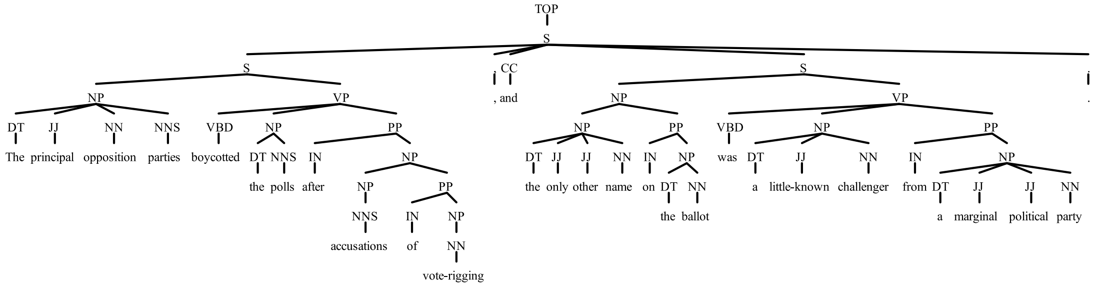
TOP
S
S CC S
NP VP , and NP VP .
DT JJ NN NNS VBD NP PP NP PP VBD NP PP
The principal opposition parties boycotted DT NNS IN NP DT JJ JJ NN IN NP was DT JJ NN IN NP
the polls after NP PP the only other name on DT NN a little-known challenger from DT JJ JJ NN
NNS IN NP the ballot a marginal political party
accusations of NN
vote-rigging

Figure 3. An Example Sentence Parse Tree

("*The principal opposition parties*", "*boycotted*", "*the polls*")
("*The principal opposition parties*", "*boycotted*", "*the polls after accusations*")
("*The principal opposition parties*", "*boycotted*", "*the polls after accusations of vote rigging*")
("*The only other name on the ballot*", "*was*", "*a little known challenger*")
("*The only other name on the ballot*", "*was*", "*a little known challenger from a marginal political party*")

The above extractions are labelled correctly in the ReVerb dataset and contain some redundant extractions. We evaluated the parser extraction on the ClueWeb12 dataset and were able to extract more than 50,000 triples. We found that the parser was able to perform on par with ReVerb and ClausIE. This was achieved using the syntactic functionality in the parse tree - Figure 3. It demonstrates the ability of a parser based approach in extracting high quality triples.

We verified the extractions for a sample set of sentences in the OpenIE and ClausIE publications. These were used to ensure precision in the parser extractions. We additionally ran the parser on the ClueWeb data and compared the number of extractions with the alternative approaches. We measured the distribution of the noun and verb sub-trees in the sentence text - Table 1. We found that 10% of the phrases were prepositional. The density of the noun and verb phrases are in agreement with the English context free grammar (CFG).

Table 1. Phrase Distribution

Noun		Frequency
NP	→ NN	14%
NP	→ NP PP	12%
NP	→ DT NN	12%
NP	→ NN NN	6%

Verb		Frequency
VP	→ VB NP	16%
VP	→ VB VP	10%
VP	→ TO VP	9%
VP	→ VB PP	8%
VP	→ VB	6%

Preposition		Frequency
PP	→ IN NP	81%
PP	→ TO NP	9%

Earlier works like OpenIE and ReVerb have looked at the extraction of subject-verb-object (SVO) triples. They were however based primarily on the availability of POS and chunker data. Structure of the verb and noun phrases were determined using statistical distribution of the phrases in text data. ClausIE used a dependency parser in resolving the SVO relations.

Projects like DBpedia [5] were designed to extract structured data in the information box and map it to an ontology. Tgrep2 [6] enable us to extract and parse a tree without explicitly coding the rules. A set of regular expressions are used to extract matching sub-trees.

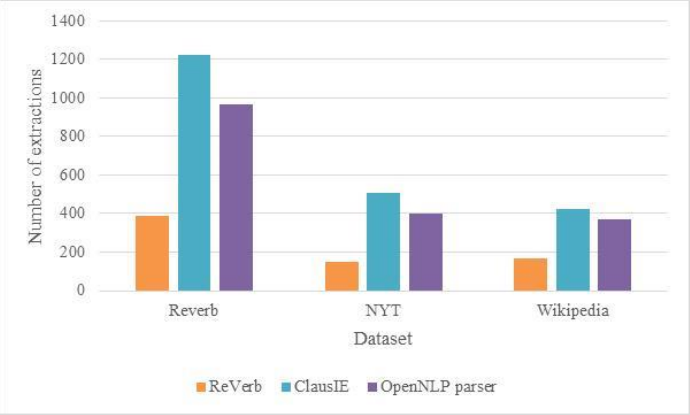

Figure 4. Number of Correct Non-redundant Extractions

We evaluated a number of extractions on the ReVerb, Wikipedia and NYT dataset. We obtained the sample dataset from the ClausIE sources. We were able to extract more than 2000 SVO in the dataset with 1000 matching the ClausIE extractions.

As all the extracted results are semantically accurate, the precision of the results is ~0.9. This value is independent of the dataset and is derived from the extraction grammar rules. The extractions are based on a rule based system and capture the syntax of the English language. Some of the SVO outputs are incorrect due to the ambiguities in the language parse tree including conjunctions in noun phrases. We verified the extracted triples to measure the recall of the data. The recall value is a function of the grammar. We can refine the rules to find additional triples in the data. This would increase the recall on the extracted values. We measured an average recall value of 60% on the triples - Table 2. We used the extractions-all-labeled as a baseline for our computation. These include all the extractions from ReVerb, ClausIE and other OIE utilities.

We estimated a precision of 0.8 for the parser extractions. We found that the parser was able to extract 2-2.5 times the correct extractions of ReVerb and 80% of the correct non-redundant ClausIE extractions - **Error! Reference source not found.**.

Table 2. Precision and Recall Values for Various Datasets

	Precision	Recall
NYT	0.8	0.64
Wikipedia	0.8	0.71
ReVerb	0.8	0.53

4. CONCLUSION

We presented a methodology for extraction of subject-predicate-object triples in a text corpus. We plan to extend this work to a larger ontological engineering for knowledge inference. We found that a syntactic parser was able to accurately extract triples in a text. We explored opportunities to further extend this work in translating an unstructured corpus of data into a semantic ontology. A user is able to explore the text using a triples structure.

Provide a statement that what is expected, as stated in the "Introduction" chapter can ultimately result in "Results and Discussion" chapter, so there is compatibility. Moreover, it can also be added the prospect of the development of research results and application prospects of further studies into the next (based on result and discussion).

REFERENCES

[1] Etzioni O, Fader A, Christensen J, Soderland S, Mausam M. *Open Information Extraction: the Second Generation*. Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, 2011.

[2] Corro LD, Gemulla R, *ClausIE: Clause-Based Open Information Extraction.* Proceedings of the 22nd International Conference on World Wide Web, 2013.

[3] OpenNLP, see https://opennlp.apache.org.

[4] Hopcroft J, Ullman J, Introduction to automata theory, languages, and computation. Addison-Wesley, 1979.

[5] Auer S, Bizer C, Kobilarov G, Lehmann J, Cyganiak R, Ives Z, DBpedia: A nucleus for a web of open data. In The Semantic Web. 2007; 4825: 722-735, *Springer*.

[6] Tgrep2, see http://tedlab.mit.edu/~dr/Tgrep2.

BIOGRAPHY OF AUTHOR

Shaun D'Souza obtained a M.S.E. degree in Electrical Engineering from the University of Michigan, Ann Arbor and a B.S. degree in Computer Science, Electrical and Computer Engineering from Cornell University. He is currently working as a Technical Lead in the CTO Office at Wipro. His research interests include machine learning, compilers, algorithms and systems.