Domain Independent Information Extraction Project Report

Gowtham Goli
N17656180
sgg308@nyu.edu
Dept. of CS

Chakshu Sardana
N10020654
cs4511@nyu.edu
Dept. of CS

New York University

1st May, 2016

Abstract

Information extraction (IE) is the task of automatically extracting structured information from unstructured and/or semi-structured machine-readable documents available on the web. This paper discusses our implementation of Open Information Extraction from the Web [4], a new extraction paradigm where the system makes a single data-driven pass over its corpus and extracts a large set of relational tuples without requiring any human input. Our methodology primarily consists of four important modules *Relation Extraction*, *Building Classifier*, *Single Pass Extractor*, *Query Module*. Using Stanford Parser's dependency graph we form tuples. A Naive Bayes classifier is trained to label tuples extracted from a corpus as either trustworthy or untrustworthy. During querying, all similar tuples with a certain overlap are grouped together and ranked. The tuple with the highest rank is returned as the answer to the query. We then report our experiments on different domain-diverse paragraphs taken from various sources over the web.

1 Introduction

Information is hidden in vast amount of data that is available now. In Information Extraction, the goal basically is to extract from the documents salient facts about specified types of events, entities or relationships. Information extraction benefits many text/web applications, for example, integration of product information from various websites, question answering, contact information search, finding the proteins mentioned in a biomedical journal article, and removal of the noisy data [6]. Furthermore it can constitute a core component technology in many other NLP applications, such as Machine Translation, Question Answering, Text summarization, Opinion Mining, etc. Information Extraction is a challenging task; some of the reasons being: use of pronouns like he/she/they etc. make it difficult to analyze, there are many ways of expressing the same fact, and so on. Hence, Information Extraction has traditionally relied on extensive human involvement in the form of hand-crafted extraction rules or hand-tagged training examples and is generally aimed at specific domains like newswire stories, seminar announcements etc. These type of approaches can not work where the number of target relations is very large, or where the target relations cannot be specified in advance. Our approach, which is unsupervised learning based, relies on only unannotated data with no fixed relations

which makes it domain independent. This paper discusses our approach to implementing TEXTRUNNER, a fully implemented Open IE system [4], and demonstrates its ability to extract massive amounts of high-quality information from a large Web page corpus, our results and potential future work. Finding semantic relationships between different parts of a sentence is one of the key tasks of Information Extraction. From the corpus, we extract relation tuples of the form *(Subject, Verb, Object)* where Subject and Object are noun phrases. Based on the tuples extracted, queries are answered which is described in detail in the following sections.

2 Approach

Our approach mainly consists of four important phases

2.1 Relation Extraction

In this phase, we parse through the dataset and extract relation tuples from each sentence. Extractions take the form of a tuple $t = (e_i, r_{ij}, e_j)$ where $i < j$ and e_i and e_j are entity strings meant to be the subject and object in the sentence and r_{ij} is a string meant to be the the relationship between them. We assumed e_i and e_j to be nouns, $e_i, e_j \in$ *(NNP, NN, NNPS, NNS, CD)* and r_{ij} to be verbs, $r_{ij} \in$ *(VBN, VBP, VB, VBD, VBG, VBZ)*. We used the Jython interace for Stanford Parser [5] to find all the base nouns and verbs in each sentence. For each noun pairs of the form (e_i, e_j) such that $i < j$, we added all the verbs contained in the sentence thus forming tuples of the form (e_i, r_{ij}, e_j).

Not all of these tuples extracted are guaranteed to hold a valid relationship. So the next step of this phase is to classify the extracted tuples as trustworthy or untrustworthy. To do this, we used the Stanford Parser's dependency graph with collapsed dependencies (the prepositions are added into the relations). The dependency graph for the sentence *Messi has won seven La Liga titles and four UEFA Champions League titles, as well as three Copa del Rey titles.* is shown in Figure 1

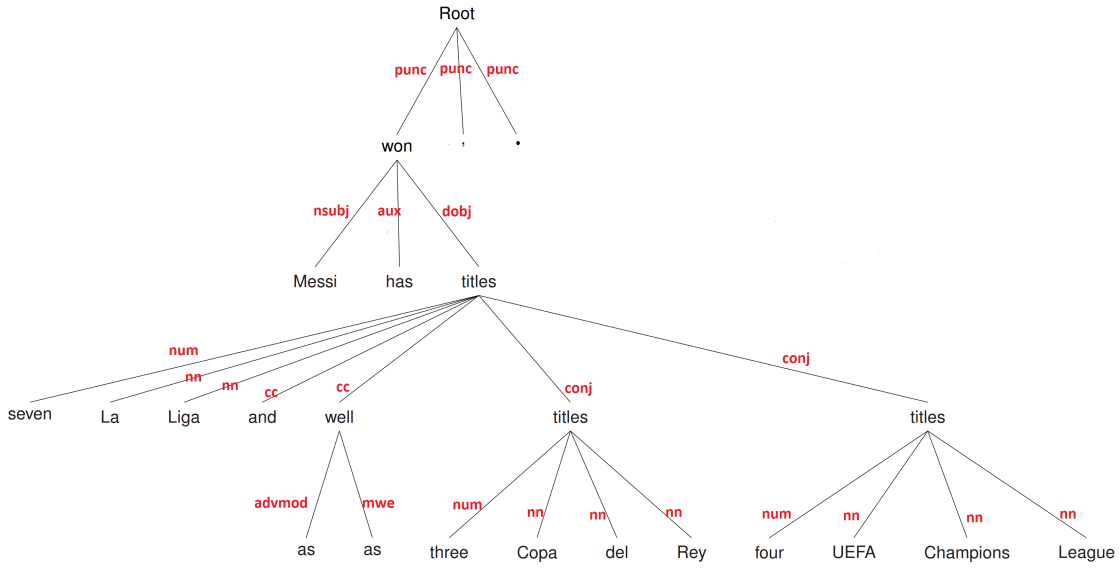

Figure 1: Dependency Graph

Using this dependency graph, we classified tuples as trustworthy if they meet certain constraints and untrustworthy otherwise. The following are the constraints we used

- There exists a dependency chain between e_i and e_j that is no longer than a certain threshold value
- Neither e_i or e_j consists solely of a pronoun.

We used Dijkstra's algorithm [1] to determine the shortest path between any two nodes in the dependency graph Few trustworthy and untrustworthy tuples of the above sentence using a threshold value of 2 are shown in the Figure 2

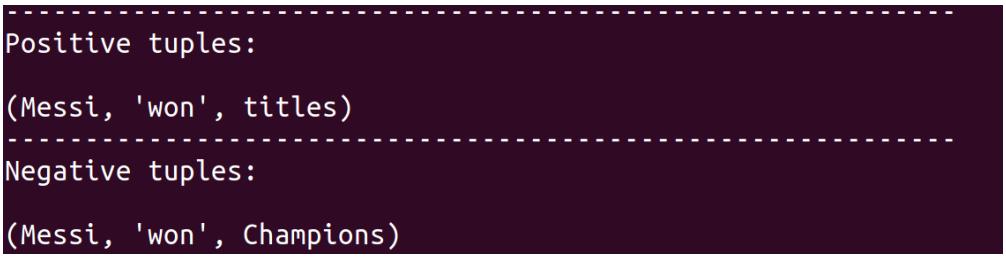

Figure 2: Trustworthy and Untrustworthy tuples

As it can be observed that there isn't much information being retained in the tuples after extraction, like what titles Messi has won, how many titles he has won etc. To overcome this, we've included the noun phrases instead of just using nouns. Each entity is mapped to the noun phrase it belongs to. If there are multiples noun phrases an entity belongs to, it is mapped to each of the noun phrase. If e_i and e_j belong to the same noun phrase, then they are not mapped. The resulting tuples after this mapping is shown in the Figure 3. It can be observed that the tuples now contain more information.

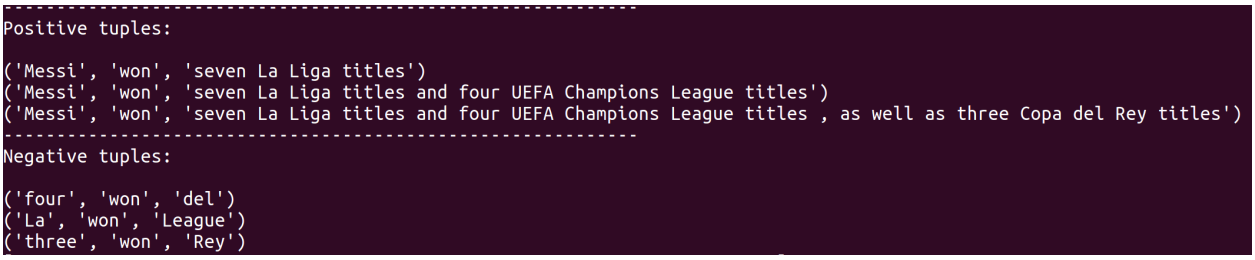

Figure 3: Trustworthy and Untrustworthy tuples with noun phrases

2.2 Building a classifier

Using Stanford parser for dependency graphs is computationally expensive and time consuming which is not suitable for real time purposes. To resolve this, we need to build a classifier that will label a given tuple as trustworthy or untrustworthy. For this purpose, every tuple extracted and labeled as either trustworthy or untrustworthy in *Relation Extraction* is mapped to a feature vector representation, which will be used to build a classifier. All features are domain independent and can be evaluated at the extraction time without the use of a parser. The following are the features included

- Presence of POS tag sequences in the relation r_{ij}
- The number of tokens in r_{ij}
- The number of stopwords in r_{ij}
- Whether or not an object e is found to be a proper noun
- The part-of-speech tag to the left of e_i
- The part-of-speech tag to the right of e_j
- Distance between e_i and r_{ij} in the sentence
- Distance between r_{ij} and e_j in the sentence
- Distance between e_i and r_j in the sentence

All the above feature vectors obtained for these tuples are then used to train a Naive Bayes classifier.

2.3 Single Pass extractor

We perform a single parse over the entire corpus. Using NLTK parse trees [3], we extract noun phrases from each sentence, forming entity tuples of the form (e_i, e_j) where $i < j$. Now all the verbs within the sentence are extracted using their POS tags and are inserted between the entity tuples (e_i, e_j) to form entity-relation tuples (e_i, r_{ij}, e_j). All such entity-relation tuples are extracted from the corpus. These are now sent as input to the Naive Bayes classifier modeled in the above phase. The classifier labels these tuples as either trustworthy or untrustworthy. Only the trustworthy tuples are retained and the untrustworthy ones are discarded.

2.4 Query Module

Due to time constraints, we could only partially implement the *Single Pass extractor* module. For the project to serve as an end to end system, for querying, we retained only the trustworthy tuples and discarded the untrustworthy tuples extracted from the first phase, *Relation Extraction* as suggested by our professor Dr. Grishman. For a given query, all the similar tuples are grouped together and assigned probabilities. The tuple with highest probability is returned as the answer to the query. This is described in detail below.

The query module accepts 4 types of queries.

- _ q_2 _
- q_1 q_2 _
- q_1 _ q_3
- _ q_2 q_3

where q_1, q_3 should be nouns and q_2 should be a verb

2.4.1 _ q_2 _

From the trusted tuples, only tuples of the form (e_i, r_{ij}, e_j) where $r_{ij} = q_2$ are retained. Let n be the count of all such tuples. Now from these tuples, similar instances of each entity are grouped together. Any two entities are e_{i_1} and e_{i_2} are considered to be similar if they are either sub string or super string of each other. Let $|e_i|$ be the count of similar instances of e_i and $|e_j|$ be the count of similar instances of e_j. Then

$$P((e_i, r_{ij}, e_j)) = \frac{|e_i| \times |e_j|}{n}$$

2.4.2 q_1 q_2 _ **or** q_1 _ q_3 **or** _ q_2 q_3

Consider the case q_1 q_2 _. The probabilities in the other two cases can be computed similarly. From the trusted tuples, only tuples of the form (e_i, r_{ij}, e_j) where $e_i = q_1$ and $r_{ij} = q_2$ are retained. Now from these tuples, similar instances of e_j are grouped together. Let $|e_j|$ be it's count. Then

$$P((e_i, r_{ij}, e_j)) = \frac{|e_j|}{n}$$

The tuple with the maximum probability is returned as the output of the query.

To illustrate using an example, consider the following sentence extracted from our corpus, *The name India is derived from Indus, which originates from the Old Persian word Sindhi.* The *Relation Extraction* module outputs the following trustworthy tuples for this sentence.

Trusted tuples:

('The name India', 'derived from', 'Indus , which originates from the Old Persian word Sindhi')

('The name India', 'derived from', 'Indus')

('Indus', 'originates from', 'Sindhi')

('Indus', 'originates from', 'the Old Persian word Sindhi')

A few example queries that can be formed from the sentence are *(India derived __)* or *(Indus originated __)* or *(__ originated __)* etc. Running the query *(India derived __)* is shown in Figure 4

```
manbearpig@Alienware:~/acads/NLP/project/open_ie/codes$ python quer_answer.py India derived _
[u'deduce', u'infer', u'deduct', u'derive', u'reason', u'reason_out', u'conclude', u'gain', u'obtain', u'evolve', u'educe', u'make', u'create', u'come', u'descend', u'derived']
India derived _:        The name India derived from Indus , which originates from the Old Persian word Sindhi

The name India derived from Indus: prob = 1.0
The name India derived from Indus , which originates from the Old Persian word Sindhi: prob = 1.0
```

Figure 4: Answer for Query **India derived from where?**

We used NLTK's Wordnet [2] to determine synonyms of the tokens in query, which can be seen in the first line of the output (the list of of synonyms for the word *derived* in their stemmed form). Thus, a query of the form *(India deduced __)* would also produce a similar output to the query *(India derived __)* as deduce and derive are both synonyms which is shown in the Figure 5. The second line contains the answer to the query which is *The name India derived from Indus , which originates from the old Persian word Sindhi* which is followed by all the possible answers to the query with their respective probabilities in decreasing order. In case, an answer couldn't be found, NULL is returned. If two answers have equal probabilities, we considered the answer that contains a bigger noun phrase, thus ensuring more information. For example, in the above sentence the probability for both the answers is equal which is 1.0. However, the later sentence is returned as it carries more information.

```
manbearpig@Alienware:~/acads/NLP/project/open_ie/codes$ python quer_answer.py India deduced _
[u'deduce', u'infer', u'deduct', u'derive', u'reason', u'reason_out', u'conclude']
India deduced _:        The name India derived from Indus , which originates from the Old Persian word Sindhi

The name India derived from Indus: prob = 1.0
The name India derived from Indus , which originates from the Old Persian word Sindhi: prob = 1.0
```

Figure 5: Answer for Query **India deduced from where?**

3 Results

We've analyzed our experimental results by extensive human involvement which was quite tedious. To do this, we extracted a few paragraphs from various sources over the web covering a wide range of domains such as Science, News, Business, Biographies, Sports, Travel etc.

From each of these paragraphs, tuples are formed using *Relation extraction* module, discarding the untrustworthy ones. However, not all the trusted tuples extracted were relevant. For example, some of the tuples lacked context and information, some of them were meaningless etc. A few such examples are shown below classified as Useful and Non-Useful trustworthy tuples.

Useful trustworthy tuples:

('The disappointing performance', 'marked', 'the third straight year in which US GDP figures have missed the estimates made by forecasters')

('infections that invade the whole body', 'disable', 'the immune system')

('scientists', 'develop', "vaccines that restore immunity in people with systemic or ' whole body ' infections")

('The US economy', 'slowed to', 'its weakest pace in two years')

('Non-housing investment', 'dropped', '5.9 pc , its largest fall since the second quarter of 2009')

('the findings', 'show', 'the capacity of the dendritic cells to display the antigens of new viruses')

("Rummenigge 's left", 'sat', 'Pep Guardiola')

('the new Dubai tower', 'feature', 'a slender , streamlined structure')

('Lead researcher Dr Jose Villadangos , an immunologist from the Walter and Eliza Institute of Medical Research in Melbourne ,', 'says', 'systemic infections such as malaria or sepsis , a bacterial infection of the blood , overstimulate dendritic cells')

Non-Useful trustworthy tuples:

('Tesla', 'lived in', 'a series of New York hotels')

('Tesla', 'lived through', 'his retirement')

('The economy', 'grew by', 'just 0.1 pc')

('The economy', 'grew in', 'the first quarter of the year')

('the post-match banquet in Madrid on Wednesday', 'Karl-Heinz', 'night')

As it can be observed, the Useful tuples contained both context and information.

However, the Non-Useful tuples lacked either context or information. For example, *('Tesla', 'lived in', 'a series of New York hotels')* or *('The economy', 'grew by', 'just 0.1 pc')* don't provide any information about when Tesla lived in a series of New York hotels and when the economy grew by 0.1%. One way to resolve this could be by merging two or more Non-useful tuples so context and information can be improved. For example, *('Tesla', 'lived in', 'a series of*

New York hotels') and *('Tesla', 'lived through', 'his retirement')* can be merged to form *('Tesla', 'lived in', 'a series of New York hotels though his retirement')* resulting in a Useful tuple. Similarly, *('The economy', 'grew by', 'just 0.1 pc')* and *('The economy', 'grew in', 'the first quarter of the year')* can be merged to form *('The economy', 'grew by', 'just 0.1 pc in the first quarter of the year')*. Some of the trustworthy tuples were completely worthless due to wrong POS tagging. For example, *('the post-match banquet in Madrid on Wednesday', 'Karl-Heinz', 'night')* doesn't make any sense as *Kar-Heinz* is incorrectly tagged as a verb.

Once the tuples are extracted, all possible queries of the form described in *Query Module* were manually constructed from each paragraph. This amounted to 646 queries. Each of these queries is then sent to the Query module as an input. Each answer returned by the Query module was checked against the corpus manually to check for it's validity. 433 out of 646 queries were correctly answered resulting in an accuracy of 67%, (using threshold value of 2, in the dependency graph) . An example of running query *(__ detect __)* is shown in the figure 6

```
manbearpig@Alienware:~/acads/NLP/project/open_ie/codes$ python quer_answer.py _ detect _
[u'detect', u'observe', u'find', u'discover', u'notice', u'spy', u'sight']
_ detect _:     Policymakers at the central bank noted in a statement released on Wednesday

Policymakers noted in a statement released on Wednesday: prob = 0.25
Policymakers at the central bank noted in a statement: prob = 0.25
Policymakers noted in a statement: prob = 0.25
Policymakers at the central bank noted in a statement released on Wednesday: prob = 0.25
specialised sentries detect a virus or bacteria: prob = 0.111111111111
specialised sentries of the immune system , dendritic cells , detect a virus: prob = 0.111111111111
specialised sentries detect a virus: prob = 0.111111111111
specialised sentries of the immune system , dendritic cells , detect a virus or bacteria: prob = 0.111111111111
Policymakers noted slower economic growth: prob = 0.0833333333333
Policymakers at the central bank noted slower economic growth: prob = 0.0833333333333
slower economic growth noted in a statement released on Wednesday: prob = 0.0833333333333
slower economic growth noted in a statement: prob = 0.0833333333333
```

Figure 6: Answer for Query **What detects what?**

As seen in the figure, the highest ranked answer returned is *Policymakers at the central bank noted in a statement released on Wednesday* (since *notice* and *detect* are synonyms) with a probability of 0.25. In our evaluation, we've marked all such answers as correct. Some of the lesser probable answers returned are a) *specialised sentries of the immune system , dendritic cells , detect a virus or bacteria* b) *Policymakers at the central bank noted slower economic growth* c) *slower economic growth noted in a statement released on Wednesday*.

4 Future Work

In future, we plan to implement capturing context and information from Non-Useful tuples as described above. It is also possible to make logical references using two or more tuples. For example, *('mice', 'injected with', 'a live vaccine')* + *('a live vaccine', 'made of', 'dendritic cells')* + *('dendritic cells', 'exposed to', 'a secondary virus in the laboratory')* $\implies$ *mice injected with a live vaccine made of dendritic cells exposed to a secondary virus in a laboratory*.

Our current corpus is relatively small which isn't good enough to train a robust classifier. We plan to add more training data including corpus from various other domains in addition to our current corpus and include additional features for the classifier. As we have only partially implemented the *Single Pass Extractor*, we plan to integrate this module into the current pipeline so that we can extend our project into a fully functional end to end query answering system.

5 Acknowledgements

We would like to thank our supervisor, Prof. Ralph Grishman, for the patient guidance and advice he has provided throughout the project.

References

[1] Dijkstra's algorithm to find shortest paths. https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm.

[2] How to use nltk's wordent. http://www.nltk.org/howto/wordnet.html.

[3] Nltk parse trees. http://www.nltk.org/book/ch08.html/.

[4] Michele Banko, Michael J. Cafarella, Stephen Soderland, Matt Broadhead, and Oren Etzioni. Open information extraction from the web. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 2670–2676, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[5] Viktor Pekar. Jython libraries for stanford parser. https://github.com/vpekar/stanford-parser-in-jython/.

[6] Jie Tang, Mingcai Hong, Duo Zhang, Bangyong Liang, and Juanzi Li. Information extraction: Methodologies and applications.