Abstract

An extremely large number of Web pages in the Internet contain free texts in natural language that are only read by human beings. To be understood by machines, these pages should be annotated with semantic markups. Manually annotating large number of pages is an arduous work. This has made automatic semantic annotation an urgent challenge. In this paper, we propose machine learning based automatic annotation approach. This approach can be trained for different domains and requires nearly no manual rules.

1. Introduction

Presently any person interested in acquiring domain knowledge approaches either an expert source or uses a search engine to track down web documents most of which exist in natural text format i.e.; in HTML format that is fully human understandable instead of machines. Although different search engines apply different ranking techniques to sort, place and return the hits, the result is always in the form of a list of handful of HTML documents the information from which has to be assimilated and understood by the user. Thus the understanding of the knowledge is in fact limited by the capabilities of the assimilator himself with little guarantee that the interpretation is correct. Further, the person has to have some preexisting ideas on the subject himself to decide the relevancy of the documents returned as hits. If these natural text documents could be annotated with semantic markup, in a way that machines can understand and process them automatically, then knowledge sharing and exchange of information in a particular domain over the Web will be facilitated. In this paper we propose to design a framework where natural text from the World Wide Web, in an automated fashion with almost no supervision.

2. Related Work

Sergey Brin laid down the concept of a learning pattern matcher in his paper [1], where a few seed phrases having a known relation were used in locating the various phrases that conveyed the same information. The newly obtained phrases would then be pruned of potentially incorrect ones, and those that remained would be added to the list of possible patterns that conveyed the same concepts as the original pattern. These would then be used to acquire new information to enrich the data warehouse. Brin's idea was taken further in the Snowball system which is a language and a compiler system where the rules of stemming algorithms can be expressed in a natural way and acting as a language-specific tool in an Information Retrieval (IR) system. Using the GATE (General Architecture for Text Engineering) framework similar modules implementing NLU has already been implemented. The most complete work in our view is that of KIM (Knowledge and Information Management) with its OntoText concept [4][5].

3. Our Approach

We designed a framework that will understand the semantics of natural text from the World Wide Web, and store it in a manner that is compatible with most machine based understanding systems for the Semantic Web that are usually large ontologies. This will also allow us to use agents to decompose user queries into semantically similar queries for ontologies, thus making the process transparent to the end user. Hence, we will be able to return the best possible information set to the user, without him having to manually do the annotating process, just because it has already been done for him by our framework.

4. Proposed System

The architecture of our proposed system is shown in Figure 1.
The Web pages are first stripped of the HTML markup, especially those that do not contribute to discovering the semantics of a document. After this process we obtain natural text corpora that we proceed to pass on to the NLP Module for further processing. The NLP module process the text through four layers: Sentence Delimiter (Detects each sentence separately); Tokenizer (Assigns lexemes with their correct tokens that help the sentence tokenizer to disambiguate between cases where it might not have sufficient information to make that decision by itself); Part of Speech (POS) Tagger (Assigns tags to the words corresponding to the various parts of speech they represent); Named Entity Finder (Recognizes proper nouns in natural text and further, classifies them as belonging to types from people, organizations, places, companies, money and so forth). The module that handles this is called the NameFinder. Using the POS Tagger and Named Entity Finder, at first the entities in the text are detected and typical patterns are looked for between the entities. For the corpus “Steve Jobs made it clear that Microsoft products were never going to have the class and style that Apple comes up with. Bill Gates refused to comment on Apple”, the entities detected are:
P1: <person>Steve Jobs</person>
P2: <person>Bill Gates</person>
O1: <organization>Microsoft</organization>
O2: <organization>Apple</organization>

Once POS Tagging and NameFinding have concluded successfully, we proceed to the next pass - normalizing the text. Normalization consists of co-reference relation resolution and preprocessing to annotate the text for later stages. We employ Co-reference relation resolution, better known as Anaphora Resolution in replacing all pronouns (he, she, it) by the entity name or ID assigned to this entity. For example, “Bill is 20 years old. He went to school” will be normalized to “Bill is 20 years old. Bill went to school”. This way Hyponym (“is a” relationship), Holonym / Meronym (“has a” relationship) extraction would be simplified. Next a relationship tree annotated with the information obtained via the above principle is built up. Currently, RDF is being employed to serialize the relations that are obtained by walking the tree [3]. Parse failures on a certain text is logged and later inspected by an expert (which maybe a human having domain expertise or a suitable agent possessing the required knowledge) to pinpoint the reason for the error. The process of information extraction is very novel. The framework first begins by validating the HTML of the captured page for well-formed ness. On passing this test, the NameFinder is run, to decide whether the corpora can afford us meaningful relationships.

Example One: Let us consider a corpus, which, for most cases do not have a possibility of yielding useful relationship among classes as depicted in Figure 2.

```
Fig.2. A List of Names without Inter-relationships
```

We now know that we have obtained a list of names, but how are they related? Do they work for the same organization? Are all their ages same? Were they born on the same date? To decide whether the information thus obtained might yield a set of relationships, we employ a heuristic function to make the decision for us. From the classes we are currently recognizing for example: Person, Organization, Money, Location, etc., the presence of the following classes might suggest the possibility of discovering relationship between them: a) organization, person; b) money, person; c) money, organization; d) location, person etc. However we cannot outright, reject a list of names, assuming no possibility of conveying relationships among them. For example, we might have:

```
Subhobroto is the friend of Sourav
```

```
Fig.3. A List of Names with Interrelationships
```

Apparently, as highlighted in this case shown by Figure 3, a list of names does embody a set of interrelationships.

Example Two: Now let us consider a corpus containing possible relationships shown by Figure 4, which takes us to Example Two.

```
Fig.4. Output of NameFinder
```

The “NameFinder” output affords the possibility of a pattern that may yield one or more relationship, extracting which requires us to build a dependency tree between entities. To do this, we feed the text into “POS Tagger”, to get the following output depicted by Figure 5:

```
Fig.5. Output of POS Tagger
```

This makes it apparent from the VBN that a relationship exists between the entities of type person and date,
allowing us to generate the dependency graph as shown in Figure 7. But before that we look up the Synset for synonym resolution to find out possible synonym sets of the corresponding VBNs. Here the possible synsets returned for the VBN born by the look up results are as captured in Figure 6 [6][7]:

Fig.6. Possible Synsets of born

These synsets are matched with the attributes of the respective NNPs represented as schemas, and the closest match found is used to fill up the attribute values. In this case, birth is found to be an appropriate solution for the purpose of resolving the VBN born, to populate the RDF Schema for a person.

Fig.7. Dependency Graph

Serializing this dependency tree into RDF yields the format as shown by the following Figure 8:

Fig.8. Format of RDF Schema

4.1 Process Algorithm

Here the algorithm to process the steps described in the architecture is presented:

INPUT: A Web page (P), written in natural text or HTML.
OUTPUT: The serialized or RDF format of the processed data of (P).

Step1 Parse the natural text corpora.
Step2 IF the parsing is successful, THEN GOTO Step5.
Step3 Log unknown class or classes which throw parsing exceptions to log.
Step4 Process automatically or notify expert to handle the unknown classes or exceptions. GOTO Step8.
Step5 Normalize the parse tree.
Step6 Process and tag or identify and correlate entities recognized.
Step7 Serialize the processed data to RDF.
Step8 End.

5. Experimental Results

We have used Java based open NLP framework to demonstrate our idea [8]. This framework has employed different processing modules, to which we have given natural text as our input and are able to receive different outputs as shown in the screen shots below: In Figure 9, the screen shot of a typical “POS Tagger” output is shown.

Fig.9. POS Tagger output

Similarly, the processed output of the “NameFinder” is captured as shown below in Figure 10.

Fig.10. NameFinder output

6. Conclusions

This paper presented a framework which can be summarized as an “Information Extraction and data mining application employing Natural Language Understanding”, to enrich the Semantic Web with preexisting content from the normal Web that we are all acquainted with.

7. References