Text Semantic Similarity

Progress Report

5 March, 2017

Hetul Shah, Kunal Suba, Mahrash Patel, Maitrey Mehta, Mohit Vachhani

***Abstract*—This is the first progress report for the project undertaken in the subject of Machine Learning. The progress made so far is understanding the problem statement and surveying the already existing solutions for the problem. The next step taken was to understand fundamentals of Natural Language Processing as a domain and its underlying concepts and applications. This was extended by a deeper look into semantics and concepts such as 'Part of Speech Tagging'and looking at it as a supervised learning problem. In the last section, we show a rough implementation of a similarity checker(lexical).**

I. Understanding the Problem Statement

Natural Language Processing is a broad domain which is an umbrella that covers various sub-domains such as Text Mining, Analytics, Information Retrieval, Text Summarization etc. Text similarity is a relatively unexplored problem in the subdomain of Text Mining. In general, text similarity encapsulates the measures of closeness between any two pieces of text. Fundamentally, the measure of closeness can be determined in two ways, one, by the meaning of the sentence, and two, by the surface closeness or the word by word similarity. The former is regarded as 'semantic similarity' while the later is known as 'lexical similarity'. Solutions to lexical similarity measures have already been achieved. One possible implementation of a solution is covered in the last section of the report. Lexical similarity checkers are easy to implement but cannot be used for applications involving semantic boundaries. For example, consider the sentences: "Maitrey helped Mohit" and "Mohit helped Maitrey". These sentences would be a hundred percent similar if a lexical similarity checker is used although it is evident that the sentences are semantically completely different. Lexical similarity checkers, however, provide a base for building semantic similarity checkers.

II. Similarity Checkers

A. SEMILAR

SEMILAR(SEMantic simiLARity software toolkit) is one application that infers the meaning of a target text and compares to that of another test text and measures the similarity. The software uses a range of methods to measure the similarity from simple lexical overlap approach to sophisticated methods such as LSA(Latent Semantic Analysis). The underlying algorithm uses Stanford Core NLP Library for tokenization. SEMILAR Corpus complements an already existing Microsoft Research Paraphrase Corpus(MRPC) by providing human-made annotations, describing relations between lexical tokens or phrases in the two texts that form a paraphrase instance in MSRP. The tokens/phrases of both the texts are compared and categorized into five different relation types according to their relative closeness in meaning. The relation types are: Identical, Close, Related, Context, Knowledge and None.

B. MOSS

MOSS(Measure Of Software Similarity) determines the similarity in programs. The main application of MOSS has been to check plagiarism in softwares. It can compare codes in 26 programming languages. MOSS uses local document fingerprinting algorithm 'winnowing' for checking semantic similarity. More about document similarity is given in [1].

C. Turnitin

Turnitin is an academic tool to check similarity between a piece of document submitted by a student and a repository of doxuments from different sources. Currently, the repositories contain three primary databases: 1) An extensively archived copy of publically accessible Internet. 2) Commercial pages from books, newspapers and journals and 3)Student papers already submitted to Turnitin.

III. Natural Language Content Analysis

Natural language Content Analysis is branched out into four fundamental types of analysis. Consider a sentence, "A dog is chasing a boy on the ground". Here we label each word with its corresponding type in English grammar. Therefore, "A" will be labelled as an article, "dog" as a noun, "is" as an auxillary verb and so on. This is part of speech tagging or lexical analysis. There are two types of constraints while tagging:

1) Local: The word 'can' is more likely to be a modal verb than a noun.
2) Contextual. A noun is much more likely to follow a determiner(article) than a verb.

These can then be combined to give phrases. For example, "A dog" becomes a noun phrase, "is chasing" becomes a complex verb and so forth. This is syntactic analysis and is used to establish the structure of the sentence. Semantic analysis is

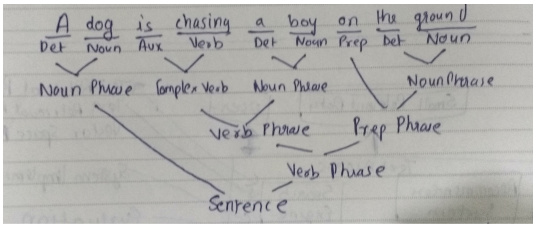

Fig. 1. Part of Speech Tagging

performed to infer the meaning of the sentence. The fourth kind of analysis is pragmatic analysis or SpeechAct which is about asking why a person made the statement or what is the goal behind making the statement. It also involves making additional inferences based on the semantic analysis made. For example, in the above mentioned statement the semantic analysis would draw out the meaning of the sentence based on which some basic inferences could be made such as " the dog is chasing the boy, hence, the boy is scared".
However, these analysis is not simple. There are many complications during each stage. For example:

1) Word-level ambiguity. For example, the word 'design' can be both a noun and a verb. The word 'root' may have multiple meanings.
2) Syntactic ambiguity. Consider the sentence, "A man saw a boy with the telescope". The problem here is we cannot identify who actually had the telescope.
3) Anaphora Resolution. Consider the sentence "John persuades Bill to buy a TV for himself". Here, the problem is whether himself refers to John or Bill.
4) Presupposition. Consider the statement "He has quit smoking". This implies he smoked before.

IV. Part of Speech Tagging as a Supervised Learning Problem

Consider the training examples, x_i and y_i for i=1,2...,m. For each i x_i is an input and y_i is its corresponding tag. The task is to learn a function f mapping inputs j to labels f(j).

The most desirable way is to use conditional models. Conditional models involve two steps:

1) Learn a probability distribution p(y—x) from the training examples
2) For any test input j, the label is defined as $f(j) = arg_y max(y|j)$

Another method used to model the tagging problem is the Log Linear Models. Here we have an input sentence $w_{[1:n]} = w_1, w_2, ...wn$ where w_i is the i^{th} word in the sentence. Similarly, we have a corresponding tag sequence $t_1, t_2, ...tn$. We use a log-linear model to define p($t_1, t_2, ...tn|w_1, w_2, ...wn$). The most likely tag sequence for a sentence $j_{[1:n]}$ would now be

$t1_{[1:n]} = arg_{t_{[1:n]}} max(t_{[1:n]}|j_{[1:n]})$

Another and the most efficient of the three is the Trigram Log-Linear Model. This model utilizes the fact that the tag of the i^{th} word will depend upon the tags of the previous words, optimally the last two tags. Hence there is a 4-tuple history associated with every tag which is $t_{i-2}, t_{i-1}, w_{[1:n],i}$ where t_{i-2}, t_{i-1} are the previous two tags. $w_{[1:n]}$ are the n words in the input sequence and i is the index of the word being tagged. Hence according to the Trigram Log-Linear Model, the probability distribution modifies as follows:
p($t_{[1:n]}|w_{[1:n]}$) = $\Pi_{j=1}^{n}$ $p(t_j|w_{[1:n],t_{j-1},t_{j-2}})$.

Note: t_{-1} and t_0 are taking as not relevant

V. Implementation of a Lexical Similarity Checker

A rough implementation of a lexical similarity checker was developed by using difflib library in Python. The program reads the two documents to be compared. The first is our primary file, known as the 'benchmark file', against which the second file is compared A counter is maintained that increments whenever two words match. At the end, the similarity is calculated by the ratio of words matched to the total number of words in the benchmark file. The results for the files are as follows:

```
1  Beijing is the capital of China.
2  Bern is that of Switzerland.
3  Accra for Ghana.
```

Fig. 2. Benchmark Text File

```
1  Accra is the capital of Ghana.
2  Beijing is that of China.
3  Bern for Switzerland.
```

Fig. 3. Test File

```
G:\ML>python p1.py
100% similarity
```

Fig. 4. Results

References

[1] S. Schleimer, D. Wilkerson and A. Aiken, *Winnowing: Local Algorithms for Document Fingerprinting*