Using multiple solutions for solving coreference resolution

Aaron Batilo
University of Colorado at Colorado Springs
Computer Science Department
Colorado Springs, CO
Email: abatilo@uccs.edu

***Abstract*—Coreference resolution is a difficult problem to solve. Despite the field of Natural Language Processing being half a century old, the field is far from solved. Coreference resolution in particular is extremely difficult and is far from being solved. This project is a critical analysis of 3 different tools that were built specifically to solve problems in the space of coreference resolution and to also explain why coreference resolution is a difficult problem.**

I. Introduction

Coreference resolution is the problem set in which the meaning behind the various forms of pronouns get attempted to be determined. This problem seems extremely trivial to humans, however, instructing a computer to solve these links between words has proven to be nearly impossible. Even with situations like those presented by the Winograd Schema Challenge, humans are known to be able to maintain an accuracy of well above 90% in determining coreferences. Many of the state of the art machine learning approaches to coreference resolution can only achieve maximums of 70% accuracy in determining coreferences.

A. Why is it so hard?

Coreference resolution is difficult largely because human's tend to poorly follow the rules of their language grammar. Hence why the field is called natural language processing, understanding the language used for communication between people doesn't follow any true set of rules. Words are spoken in the incorrect order, verb tense is inconsistent, or pronunciations/spellings of words is widely off from being correct. Any combination of these errors or others means that being able to parse natural language is inconsistent.

Humans are able to cope with these inconsistencies because we are able to correlate words to their meanings. We can actually understand the meaning behind various word combinations or sequences and so it is easy for humans to contemplate the order of words in a phrase, on top of what this specific order of words intends to mean.

II. The method

By combining different solutions with different strategies for solving coreferences, it's likely possible for us to have more accurate results in general. While it is true that this is a slower solution, what I'm focusing on is the fact that in theory, these solutions should be able to feed off of each other's results. In order to test this theory, a total of 3 different pre-prepared coreference resolution tools were tested. The corpora used was based on 20 different news articles of between 1,400 and 1,500 characters in length that were found and stored by an application written by my colleague Scott Denning. The corpora was then coreference resolved by myself by hand. To help keep my research simple, solutions were evaluated on the delta of how many coreferences there were in a given article, averaged out over the 20 articles that were provided to each system. Each solution provided its own opinion and value when determining which words formed coreference pairs, and the critique of each solution and the results provided by them are given later in this paper.

III. Stanford's CoreNLP

A. Background

Stanford University has its own group of people that tackle the NLP problem. Many publications and advancements for the field have come from the Stanford NLP group, and it is no surprise that they have bundled their findings into an open source collection of tools. This collection of tools they have named CoreNLP which is available to use under the GNU GPL v3 license. Among the tools that make up the CoreNLP suite, includes a coreference resolution module that follows the method of resolution published in the paper [1] which is a statistics based, multiple pass sieve approach to coreference resolution. It is void of any complicated machine learning models.

B. Implementation

For the sake of simplicity, an open source python wrapper for the CoreNLP library was used. This wrapper can be found *link to github*. The wrapper itself has two modes of execution. You can chose to run an application as a JSON-RPC server which will parse and return results or you can include the wrapper as a module in your own application for local usage. Due to the size of the CoreNLP suite, a large amount of resources is required to run this library. the Stanford NLP group suggests a minimum of 3gb of RAM, and the test computer is a 2015 MacBook Pro with 16gb of RAM and a 2.2 GHz Intel i7 processor.

C. Strengths

CoreNLP is impressive in what it is able to determine may have a coreference pair. Many of the examples contained more

than just standard pronoun coreferences. This includes subjects such as phone numbers or descriptions of specific time events, both of which CoreNLP was able to resolve on top of doing the pronoun resolution.

D. Weaknesses

In terms of accuracy, CoreNLP performed fairly well, however there were several hurdles that made working with CoreNLP rather unpleasant. Firstly, despite the original paper by Jurafsky et al *paper* mentioning that their multi-pass sieve approach was supposed to be fast, we found that the CoreNLP coreference resolution module was very slow to run. Another very limiting factor appeared to be that if the parser was given more than a sentence or so of text, time seemed to go up faster than linearly which is non-intuitive to the description of the multi-pass sieve. We tried parsing a single article in its entirety which is roughly 1,450 characters, and after 18 hours of run time for a single article, the CoreNLP module still had not returned with the coreferences. My solution was to feed the system one sentence at a time, which means we lose the possibility of the system detecting intersentential coreferences.

E. Sample

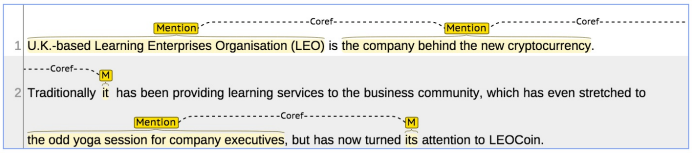

Fig. 1. Screenshot of CoreNLP demo website

IV. BART

A. Background

BART is the Beautiful Anaphora Resolution Toolkit and was developed as a result from a summer workshop back in 2007. BART is an open source solution, developed by a small team lead by Massimo Poesio. Similar to CoreNLP, BART can be used most easily by running the application as a server, however BART runs with a REST-based form of communication instead of being a JSON-RPC server. BART supports all of the steps necessary to perform the coreference resolution, and this includes tokenizing and parts of speech tagging the text as a preprocessing step. It uses multiple machine learning approaches to tackle solving coreference pairs. [3]

B. Implementation

For my project, BART was ran locally on my computer and text from my corpora was sent via a command line program named curl. We ran BART with its suggested 1gb of RAM from the JVM. Once the server was ran locally, you could either send the raw text via HTTP POST or you could use the graphic interface that was being served on your localhost.

C. Strengths

BART was by far the easiest tool to use. You could download the BART files and run it almost immediately without any problems. In terms of solving the actual coreferences, the most impressive feature of BART is the ability to solve coreferences across multiple sentences. BART is also able to link when the entities in the coreference pair are actually referring to the same thing. This is impressive in its own right specifically because these entities do not always appear the same way throughout the text. Company names or people names are often reduced down to one word but BART is still likely to know that these words are referring to the same entity.

D. Weaknesses

BART was very powerful at finding coreferences, however we noticed that there were two main mistakes that BART made. The first one being that there were multiple occassions where a company was referred to by multiple different forms of the same name, and it was also referred to as something like ”the company” and BART was unable to link the fact that ”the company” was referring to this capitalized multi-word expression. The other mistake BART often made was being unable to resolve regular pronouns as entities. Words such as ”he” or ”she” were relatively often reported as not referring to anything, despite these words being from the most common type of word to be used in a coreference pair.

E. Sample

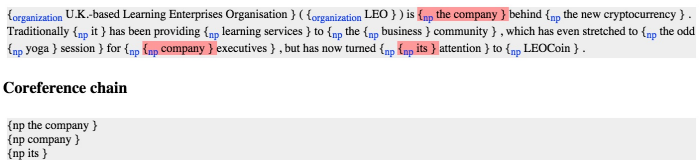

Fig. 2. Screenshot of BART web GUI

V. RECONCILE

A. Background

Reconcile is another system, just like BART, that was made entirely for coreference resolution. It was created by a group from the Computer Science department at the University of Utah as a test-bed for researchers to be able to implement new methods quickly. By default, it works out of the box and utilizes existing tools to perform the various preprocessing steps required for coreference resolution, including the Named Entity Recognizer module from Stanford’s CoreNLP engine. For the actual coreference resolution, Reconcile uses supervised machine learning classifiers that come from the Weka toolkit for machine learning. [2]

B. Implementation

Reconcile was written in Java and comes packaged in a jar file which you can run from the commandline. This was the only solution that wasn’t ran as an external server application, and instead was just ran as a commandline tool locally on the machine.

C. Strengths

Reconcile's notable strength over the other solutions was that it was able to group words together that could be used to represent an entity. Compared to the other solutions that may have only found a hand full of word groupings that could represent an entity or concept, Reconcile was able to accurately group together tens of dozens of word sequences that could represent an object or idea. Phrases such as "The more popular bitcoin" and "the latest Hero 4 cameras launch", both are a grouping of words that represent a concept. Reconcile was able to accurately find roughly 50-60 of these multi-word expressions in each of the 1400-1500 character news excerpts.

D. Weaknesses

Reconcile was by far and easily noted as the slowest tool to run. Where CoreNLP would take 4-6 seconds to run over an article, and BART would only take roughly 3 seconds, Reconcile takes roughly 12-15 seconds per article. We believe this is largely because it is passing the text between multiple tools that were packaged into the jar file such as the Berkley word parser then to the Stanford CoreNLP Named Entity Recognizer as well as accessing Princeton's WordNet. Reconcile has a lot of processes going on at once, all of which add to the total run time of the tool.

E. Sample

```
<NP NO="8" CorefID="8">
    U.K.-based Learning Enterprises Organisation (LEO
</NP>)
is
<NP NO="9" CorefID="24">
    the company behind
    <NP NO="10" CorefID="10">
        the new cryptocurrency
    </NP>
</NP>.
Traditionally
<NP NO="11" CorefID="11">
    it
</NP>
has been providing
<NP NO="12" CorefID="12">
    learning services
</NP>
to
<NP NO="13" CorefID="57">
    the business community
</NP>,
which has even stretched to
<NP NO="14" CorefID="16">
    the odd yoga session for
    <NP NO="15" CorefID="15">
        company executives
    </NP>
</NP>,
but has now turned
<NP NO="17" CorefID="17">
    <NP NO="16" CorefID="16">
        its
    </NP>
    attention
</NP> to
<NP NO="18" CorefID="27">
    LEOCoin
</NP>.
```

Fig. 3. Screenshot of highlighted Reconcile xml output

VI. Results

As was mentioned earlier, to help keep the comparison between solutions simple, the only thing that we actually compared was how many coreference pairs were found the the tool. Below is a chart of how many coreferences were found when the coreferences were solved manually by hand, along with the number of coreferences found by each respective tool that was outline above.

Article	Coreferences	CoreNLP	BART	Reconcile
1	15	13	6	5
2	4	7	5	6
3	5	15	8	8
4	8	10	6	5
5	12	17	12	7
6	3	2	10	8
7	4	5	5	3
8	7	6	10	5
9	6	9	8	6
10	5	6	10	7
11	1	3	6	6
12	11	25	11	6
13	2	3	9	6
14	4	9	10	5
15	3	12	8	8
16	9	11	13	6
17	3	3	4	3
18	3	8	3	3
19	3	7	9	9
20	4	8	6	9

TABLE I: Number of coreferences

The absolute value of the difference between the number of existing coreferences and the number found by each tool was used as the method of comparing. The deltas are shown in the following table.

Article	CoreNLP	BART	Reconcile
1	2	9	10
2	3	1	2
3	10	3	3
4	2	2	3
5	5	0	5
6	1	7	5
7	1	1	1
8	1	3	2
9	3	2	0
10	1	5	2
11	2	5	5
12	14	0	5
13	1	7	4
14	5	6	1
15	9	5	5
16	2	4	3
17	0	1	0
18	5	0	0
19	4	6	6
20	4	2	5

TABLE II: Coreference deltas

When the deltas are averaged out, we get the following results

CoreNLP	BART	Reconcile
3.75	3.45	3.35

TABLE III: Averages

This table is saying that on average, each tool will only miss roughly 3 coreference pairs. You can see that despite the differences in the approaches by each solution, they all yield very similar results. There are a few interesting notes to take away from these results. Firstly, despite having a statistics only approach, the CoreNLP tool still manages to stay very competitive with the machine learning approaches to coreference resolution. The second fact to notice is that Reconcile, which was the slowest tool to run, yeilded the most accurate results. Keep in mind that Reconcile works by combining multiple different tools to make up the preprocessing flow of its coreference resolution method. Lastly, BART was extremely close to beating Reconcile in terms of accuracy, but remember that BART also ran in approximately 1/5th the time that Reconcile did.

VII. Conclusion

A. Takeaways

A combination of tools preemptively yields promising results. In the example of Reconcile, a combination of tools allowed for a more accurate set of coreferences, at the sacrifice of execution time. we believe that with the same tradeoff, we could get a more accurate system for determining coreference pairs.

B. Next steps

Moving forward, there are several moves we can make. Firstly, we need to investigate a more precise way of determining the accuracy of different coreference resolution tools. we have already seen one named Coreference Resolution Toolkit (cort) which has both another method of resolving coreferences, on top of an error analysis component. In many papers on coreference resolution, there seems to be scores that these tools are receiving after running on test data sets. Learning how to calculate these scores will aid in this step.

The next thing we can do is implement more tools. For example getting Apache's OpenNLP engine working, or implementing cort's coreference resolution module. The more tools my suite uses, theoretically the more accurate results we will get. By having multiple tools returning the coreferences they find, my suite can return only the coreferences that appear in more than one of the tools that has been implemented.

The last step to take would be to automate this system. At the moment, each tool outputs their coreference resolution results in a different format. If we were able to create a program that acted as an intermediate layer between these tools, we could automate the process of inputting text and printing out coreference pairs. An intermedite layer is necessary anyways because these tools are written in different languages to begin with.

C. Closing

By taking this multiple tool voting approach, we believe that at the very minimum, we will be able to reduce the number of false positive coreference pairs. Perhaps if a threshold is set of N or more tools that need to report the same coreference, we can increase the confidence that each reported coreference is correct. And with more and more tools being implemented, we can get a combination of methods each with their own strengths and weaknesses.

Acknowledgment

The author would like to thank Dr. Kalita for allowing me to do an independent study through him. The author would like to thank Scott Denning who was my immediate mentor this summer while learning about natural language processing and also provided the text corpora that the author ran these tools on. Lastly, the author would like to thank my work for being flexible with my work hours so that the author could pursue this independent study.

References

[1] Heeyoung Lee, *Stanfords multi-pass sieve coreference resolution system at the conll-2011 shared task*, CoNLL (2011).

[2] Ves Stoyanov, *Coreference resolution with reconcile*, ACL (2010).

[3] Yanick Versley, *Coreference systems based on kernel methods*, 22nd International Conference on Computational Linguistics (2008).