Project Report

CS 6320: Natural Language Processing

Submitted by:
Team: Hodor

Members:
• Praveen Tangarajan
• Shariq Ali

Problem Description

Information extraction is basically defined as the task of automatically extracting structured information from unstructured and/or semi-structured machine-readable documents and other electronically represented sources. In most of the cases this activity concerns processing of human language texts by means of natural language processing (NLP). This class project aims at developing an information extraction system which extracts specified information based on the given templates.
In this project, 30 documents, 10 in each related to organization, place and persons are given and from these sets of documents, the goal is to extract the following information.
Template #1: BUY-(Buyer, Item, Price, Quantity, Source)
Template #2: WORK-(Person, Organization, Position, Location)
Template #3: PART-(Location, Location)

Proposed Solution

The solution to this project is to apply deep concepts of natural language processing to extract the required information. Initially, we started with understanding the data distribution amongst each of the three categories of the documents, basically to understand the level of cleaning and data pre-processing that may be needed.
On completing the initial data cleaning, we went on to do a lexical, syntactic and semantic analysis of the sentences within each document. This was to understand the distribution and working of the sentence grammars on these documents, with this knowledge we went on to apply different feature extraction techniques to extract crucial word/features corresponding to each of the templates, which we went on to use it as a base for out modelling of this information extraction system.
Finally, we end up developing a combination of statistical + heuristic based model for the information extraction system.

Solution Steps:

1) Understanding of the data, their structure and the distribution of words etc.
2) Cleaning – removing bad characters and symbols, spell correction etc. – used regex etc.
3) Lexical analysis
4) Tokenization – Paragraph tokenization, sentence tokenization and word tokenization.
5) Lemmatization and stop word removal – (removing stop words didn't help much)
6) Feature extraction – extracting to occurring words in document and using wordnet on these word to obtain synonyms, hypernyms, hyponyms etc.
7) Syntactic and Semantic analysis
8) Pos-Tagging.
9) Name Entity recognition (both predefined and we created custom named entities too)
10) Co-reference resolution
11) Statistical dependency parsing.
12) Modelling Heuristics/rules on the top of dependency parsing
13) Checking the correctness of the extraction and verification.-(mostly manual)
14) Results extraction and Json parsing.

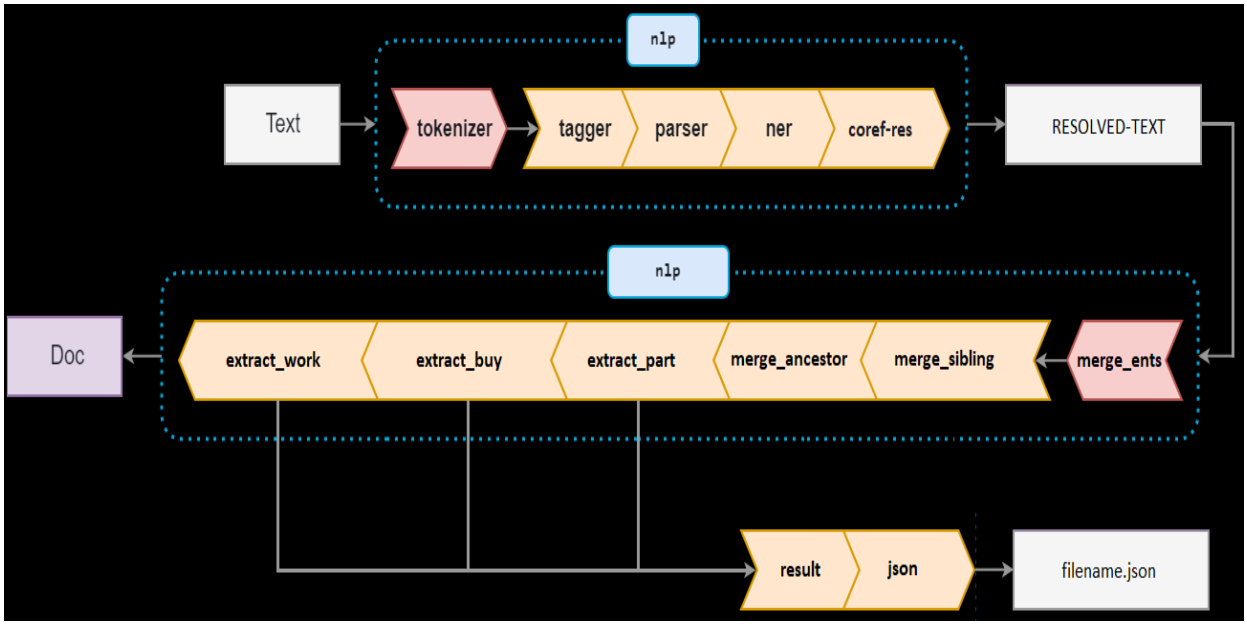
The detailed technical architecture including different frameworks and tools that are used have been mentioned the next sections.

Implementation Details

Programming languages: Python
NLP Frameworks: Spacy**,** NLTK**,** Neuralcoerf, StanfordNLP

Technical solution architecture:

Description of the solution architecture (shown in figure 1) in the following steps:



- The text from the corpus is given and processed to a series of pipelines. After initial processing the corpus is fed into tokenizer which basically does three levels of tokenization, the paragraph sentence and word tokenization. Some custom rules have been written by us to handle few cases in the tokenization part
- Then, the cleaned sentence is passed to tagger which basically does the part of speech tagging, the parser which does the dependency parsing and the ner which does the named entity recognition. Since we wanted to create few of our own entities we have created some custom named entities for our processing using entity ruler.
- Then we pass the result to a coreferece resolver, which basically resolves coreferences.
- The referenced text is then passed through a series of our custom functions like merge_ents, merge_sibilings, merge_ancestors etc. The merge_ents has rules to merge different entities as a single one for example New York is considered as two separate tokens, the function basically coverts it to a single token and recognises the whole as a place entity.
- The merge_sibilings, merge_ancestors are the functions, what they do is based on heuristic of each entity they lookout for rules by considering their sibilings and ancestors.
- Once these are extracted, the data is passed through three main functions extract_buy, extract_part and extract_work which basically extracts all the required templates based on the written grammar rules and statistics.
- Finally a json is created out of these files based on these results and saved.

Results and error analysis:

Error analysis

The logic for the buy templates is internally based on identification of a buy or its semantic form based custom created named entity instance. So on analysing the results we basically look out for the identification and correctness of the named entity tagger, the pos_tagger and the parser first.

First, we check if both the default named entities and the custom created entities are identified and recognised correctly.

Then we check the correctness of the rules written by us based on the results and the visualization of the tagger and the parser.

Once the written rules are correct, they should be able to extract the results correctly. If we are not able to get correct results we then modify it to get a better result.

Modification of the rules are done by looking at the output, if the performance of the model is to specific, that his high precision, we tend to make it more generalized by relaxing few conditions and constrains. On the other hand, if the rule is too generalized, we add more constraint to get the fall in the "just right" results area.

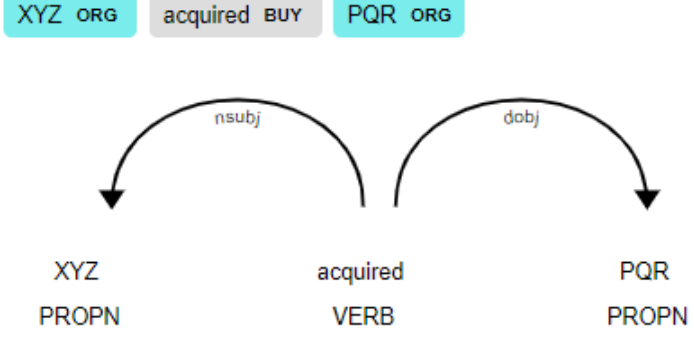
For the named entity related errors , we have tried to best handle it using heristic approach and avoided writing rule checks based on the named entity.

Consider the result analysis of the buy template:

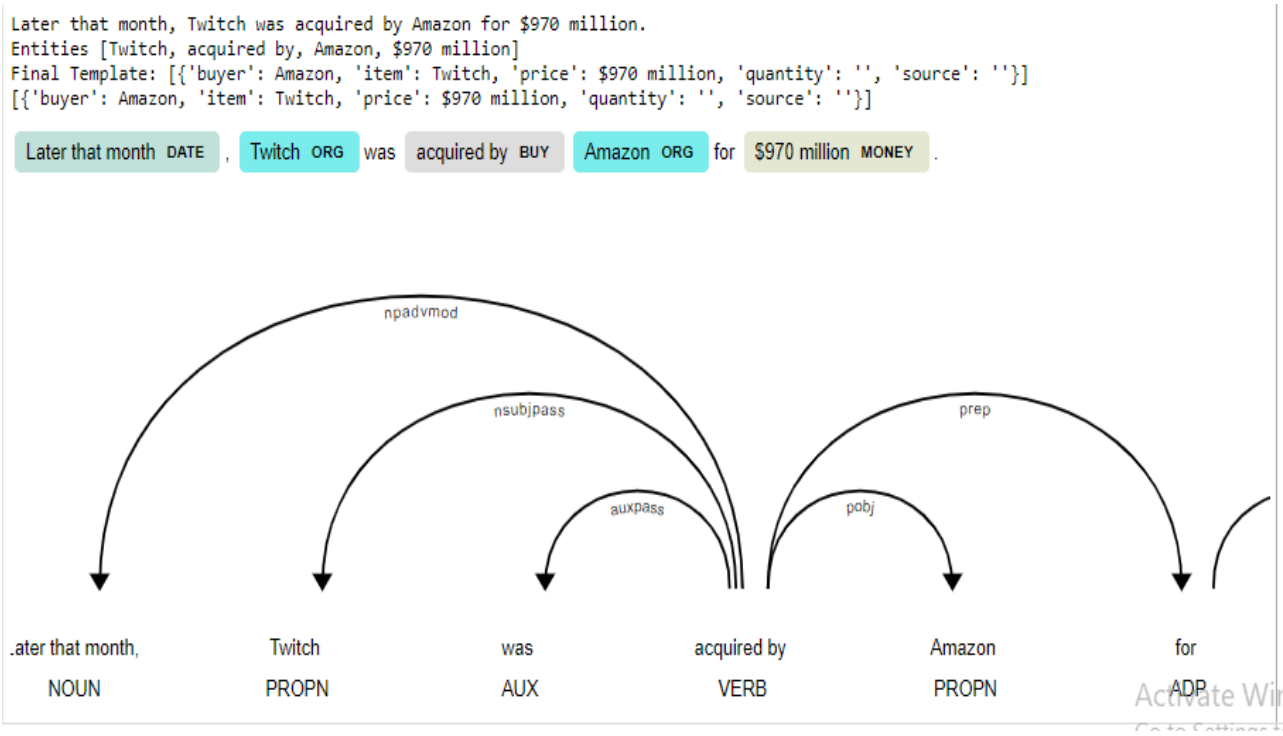
The buy template basically extracts all the instances of a buying event in the following format: buyer, item, price, quantity and source.

Consider the following results analysis of the sentence "XYZ acquired PQR":

```
XYZ acquired PQR
Entities [XYZ, acquired, PQR]
Final Template: [{'buyer': 'XYZ', 'item': 'PQR', 'price': '', 'quantity': '', 'source': ''}]
[{'buyer': 'XYZ', 'item': 'PQR', 'price': '', 'quantity': '', 'source': ''}]
```



Consider a simple sentence "XYZ acquired PQR", we can see from the results above, the first print is the sentence itself, then the second are the named entities and the final template shows the results that are extracted base on the rule " if token entity type is buy PROPN and its dependency is nsubj from the head which is a buy entity type and VERB extract it as the buyer "



Similarly a much more complex example is:

Similarly, strategy of result analysis has been done for the rest two work and place templates.

Consider the result analysis of the place template:

Sentence: Richardson is a principal city in Dallas and Collin counties in the U.S. state of Texas.

Following is the **extracted template results**:

PART(“Richardson”, “Dallas”)

PART(“Richardson”, “Collin counties”)

PART(“Richardson”, “U.S. state of Texas , Texas”)

PART(“Texas”, “U.S.”)

These were derived from the similar approach as explained in the buy part.

Consider the result analysis of the work template:

Sentence : Steven Paul Jobs (; February 24, 1955 – October 5, 2011) was an American business magnate and investor. He was the chairman, chief executive officer (CEO), and co-founder of Apple Inc.;

Following is the **extracted template results**:

WORK(“Steven Paul Jobs”, “Apple Inc.”, “chairman ; chief executive officer (CEO); co-founder”, “”)

These were derived from the similar approach as explained in the buy part.

Problems Encountered:

- Finding and searching for different Natural language processing frameworks that are good at certain tasks, for example initially we tried dependency parsing tree from NLTK package and it didn't work well, was poor in visualization too. Spacy in contrast uses different object-oriented approaches where all words and sentences are objects themselves was much better and usage friendly in most of the tasks like part of speech tags, named entity recognition and noun chunks.
- Faced challenges in sentence tokenization, some sentence tokenizes in external library didn't work well as we wanted; we had to write few rules on top of them. The sentence tokenization was not able to recognize correct breakpoints in the paragraph. Instead it considered the entire paragraph as a single sentence. This makes the parse structure and the extraction difficult. So, to avoid this issue, we created a separate tokenization format for paragraphs which considers each row as a separate sentence and wrote a certain set of rules to parse the data.
- Co-reference resolution library neuralcoref was not working well with the latest spacy and python version, we face version compatible issues, had to down grade the version in order to make the co-reference resolution work.
- For the buy template and the work template, it was difficult to obtain the job titles from the sentence. We started working on training a custom named entity recognizer but data annotation and quantity of training data needed was not feasible, so we switched to creating custom named entities using ruler and matcher of the high probable words based on data distribution.
- NER in Spacy was not working properly to our needs. Was giving wrong results at times.
- Results verifying and estimating the accuracy of the developed model was a problem, we had to look through the results manually to see if the model or the rule written in the models work well.

Pending Issues:

- Improving the accuracy of the model by writing rules/heuristics for the complex transitive or multilevel dependent unusual test cases or probably increasing the set of rules required to cover a wider range of extractions.
- Improving the performance of the named entity recognizers, probably by annotating and training the named entity model, our models performance on work template drastically depends on named entity recognizers.

Potential Improvements:

- Training custom named entity tagger and dependency parser based on probabilistic modelling
- Improving the rules.