Project Report

CS 6320: Natural Language Processing

Spring 2020

Submitted by:

Team: Legacy

Members:

- Siddhartha Kasturi (SXK180149)
- Nachiappa Palaniyappa (NXP190002)

Problem Description

Today's world is filled with articles on large variety of topics. The main aim of this project is to build an information extraction product that can that can extract important information from these articles. The following are the three templates of information that we need to obtain from the articles,

Template #1: BUY(Buyer, Item, Price, Quantity, Source)

Template #2: WORK(Person, Organization, Position, Location)

Template #3: PART(Location, Location)

Proposed Solution

We plan to use various techniques from powerful Natural Language Processing packages to obtain the sematic & syntactic information from these articles. NLP based features such as Parts-of-Speech tags, dependency parse tree, coref-resolution, custom NER, tokens, stems, lemmas, synonyms, hypernyms, hyponyms, meronyms and holonyms can be extracted from every sentence, with these NLP packages. With these features, we can create a model based on heuristic approach, to extract the information in formats of above-mentioned templates

Implementation Details

Programming tools:

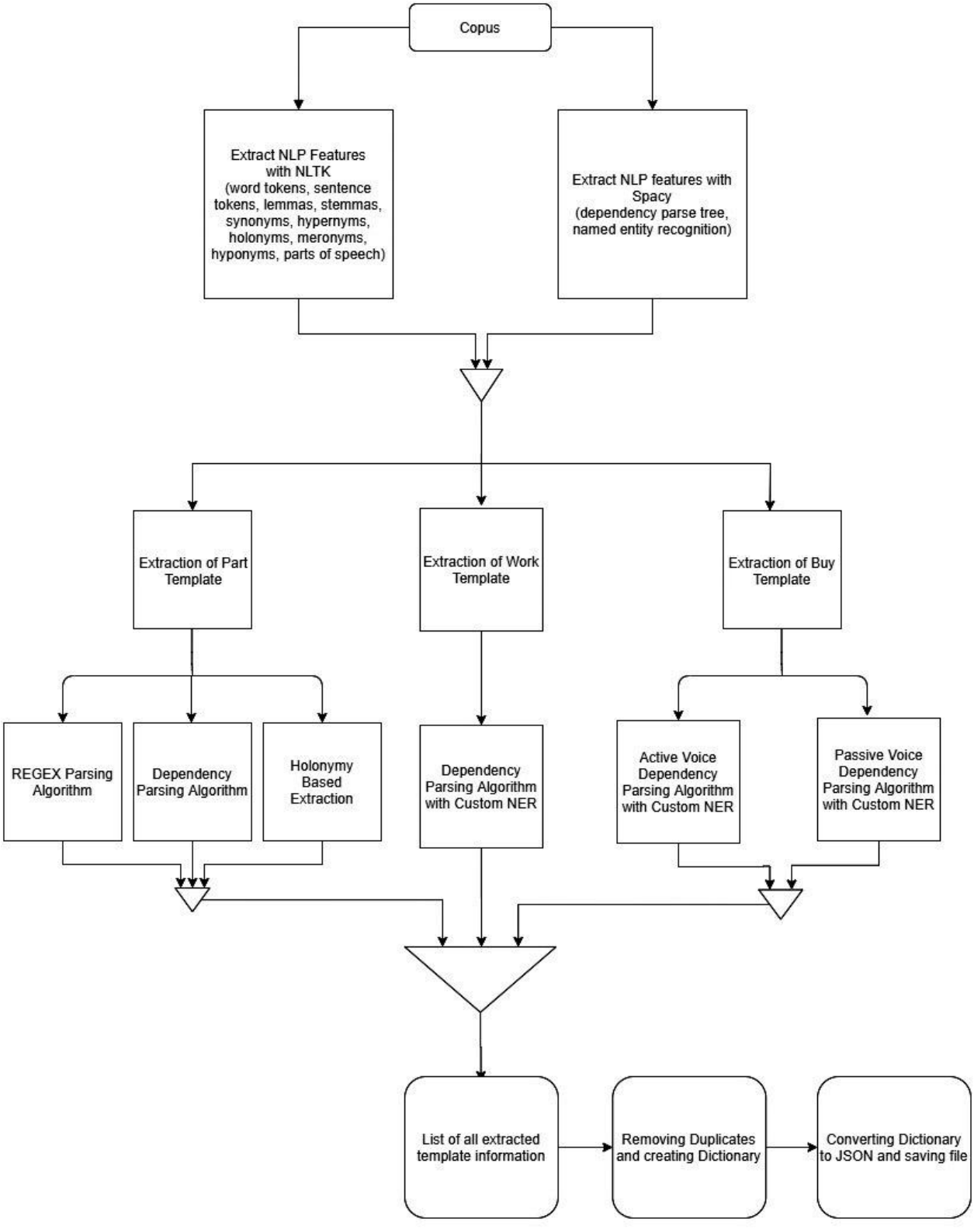
- Python (*version*: 3.7.4)
- NLTK library (*version*: 3.4.5)
- Spacy library (*version*: 2.0.1)

Architecture:

We shall cover our architecture (shown in figure 1) in the following steps:

- Read the corpus text and store it as a list of sentences. This is done with the help of Sentence Tokenization technique available in NLTK library
- Using NLTK library, NLP features such as word tokens, lemmas, stems, synonyms, hypernyms, hyponyms, holonyms, meronyms, and parts-of-speech tags are extracted from every sentence.
- Using Spacy library, a dependency parsed tree and named entities is obtained for every sentence.
- Using neuralcoref library we resolved potential co-reference problem.
- Developing custom NER for Buy activities
- Merging entities together
- Merging noun chunks
- Tokenized sentence list is iterated, and various algorithms are applied on each sentence token to extract information for the three templates.
- Removing duplicates and creating dictionary
- Converting dictionary to JSON file

Architectural Diagram:



Text Processing:

- In Spacy library package, en_core_web_sm model, which is a pretrained model for English Language is loaded
- The sentence token and sentence text is passed to the model as input to obtain container in the doc format
- The doc contains necessary information to obtain the dependency parsed tree structure. It contains a list of spans. It also contains list of all the named entities, noun_chunks present in the sentence.
- Each span represents a word token with all the necessary nlp features as position of the word in the named entity, POS, position of it in the noun chunk.
- With the help of the extacted information from the spans, we can combine the various words representing the Named entities and the noun chunks into groups of spans and make that group into a single span. Then we re-tokenize the changes into the actual doc so that it gets updated. This process is done by Spacy pipelines for Named entities and the noun chunks.
- All the documents are coref resolved using neural-coref pipe before going to the template algorithms.

BUY(Buyer, Item, Price, Quantity, Source) template:

Active Voice Dependency Parsing

- Developed custom NER to recognise transaction related activities.
- We iterate through all tokens and check whether the entity type is BUY
- Then we check its children how they are dependent on BUY entity
- If dependency between BUY and child is "nsubj" we extract it as buyer
- If dependency between BUY and child is noun and "dobj" we extract it as item and also quantity (Optional) looking at its children if entity type is “nummod”.
- If dependency between BUY and child is "dobj" we extract it as item.
- We extract money buy looking at all tokens and which is linked with that BUY activity
- We extract source when token is “from” and its children are PROPN and “pobj”

Passive Voice Dependency Parsing

- Developed custom NER to recognise transaction related activities.
- We iterate through all tokens and check whether the entity type is BUY
- Then we check its children how they are dependent on BUY entity
- If dependency between BUY and child is "pobj" we extract it as buyer
- If dependency between BUY and child is noun and " nsubjpass" we extract it as item and also quantity (Optional) looking at its children if entity type is “nummod”.

- If dependency between BUY and child is " nsubjpass" we extract it as item.
- We extract money buy looking at all tokens and which is linked with that BUY activity
- We extract source when token is “from” and its children are PROPN and “pobj”

PART(Location, Location) template:

REGEX:

- We iterate through tokens in sentence and search with entities with “GPE”
- By escaping “,” “space” “and” we club all the entities together
- If we encounter “and” we will remove all the group having connected with it
- If we find only one isolated GPE we will discard it

Dependency Parsing:

- In this approach we trace a path between two GPE or two collection of GPE’s
- If path contain certain words which are in check list that means, there is PART a relation between those entities
- We are creating two list left and right to link multiple entities
- We check various combinations like list having “and” “,” or none
- Based on combination we link them as PART

Holonymy:

- If we fail in extracting with REGEX and Dependency parsing we use Holonyms of GPE entities
- We iterate through tokens and find for token with GPE entity and check whether the sentence has its holonyms or not.

WORK (Person, Organization, Position, Location) template:

Information Extraction from Dependency Parsed Tree

- Each span has syntactic dependency relation with respect to the sentence
- We make use of this to create a certain set of rules to fill the information into the WORK template
- The named entity types required for this is given below,

Named Entity Recognition type	Format	Example
PERSON	Name of a person (noun chunk)	Abraham Thomas Lincoln
ORG	Name of an organization(noun chunk)	Google, Amazon
GPE	Name of a global location	United States of America, Dallas, Texas

Custom Named Entity Recognition:

- While PERSON, ORG, GPE are available as pre-trained entity models, we created a checklist of various POSITIONS and validate input before adding to template.
- We have obtained a list of job titles from a pre-trained model available in Stanford Core NLP of Java.

Rules for extraction:

- Named Entity Recognition play a major role in the information extraction as we can extract all the information required with just these named entities.
- But the only thing that we need to consider is that, these named entities should be linked to each other to form the WORK(Person, Organization, Position, Location) template (i.e.) PERSON ner should be linked to the ORG ner, and LOCATION ner.
- We created a set of rules to find the links between the required Named Entities to form a single WORK template information.
- Sentence token is searched for a PERSON and POSITION .If both are present, it is checked for the syntactic relation with the children of the root word in the sentence. They could also link with other entity types such as GPE, and ORG. Then, it is added to the WORK template

Information extraction from the table structured text in the corpus:

- When we checked through the input articles, we found a unique set of texts occurring in a few places. These texts were in table format and so, the sentence tokenization of the NLTK package does not work well with these data.
- A separate algorithm is written to parse these tables into tokenized sentences.
- With these sentence tokens, a few rules specific to the table formats are written to extract the information for WORK template
- The list of WORK template obtained after the iterating through all sentences is converted into json output and stored.

Results and error analysis:

The screenshot displayed below shows a sample of output format containing the extracted information.


```
{
    "document": "AppleInc.txt",
    "extraction": [
        {
            "template": "PART",
            "sentences": [
                "Apple Inc. is an American multinational technology company headquartered in Cupertino, California,
                        that designs, develops, and sells consumer electronics, computer software, and online services."
            ],
            "arguments": {
                "1": "Cupertino",
                "2": "California"
            }
        },
        {
            "template": "WORK",
            "sentences": [
                "In August 2011, Jobs resigned as CEO due to health complications, and Tim Cook became the new CEO."
            ],
            "arguments": {
                "1": "Tim Cook",
                "2": "the new CEO",
                "3": "",
                "4": ""
            }
        },
        {
            "template": "BUY",
            "sentences": [
                "In July 2001, Apple acquired Spruce Technologies, a PC DVD authoring platform, to incorporate
                        their technology into Apple's expanding portfolio of digital video projects."
            ],
            "arguments": {
                "1": "Apple",
                "2": "Spruce Technologies",
                "3": "",
                "4": "",
                "5": "July 2001"
            }
        },
```


Problems Encountered:

- Tried to create Dependency Parse Tree from NLTK package. But, NLTK is a string processing library and just returns strings. So, we made use of Spacy which has an object-oriented approach where all words and sentences are objects themselves which helps in easier access of a lot of syntactic information such as POS tags, named entity recognition and noun chunks.
- Since named entities does not recognize a position(job title), it was difficult to obtain the job titles from the sentence. Writing the rules represented by dependency parsing resulted in many false positives. To avoid this, we made use of a POSITION list which has most of the job titles. The job titles list contains a numerous amount of designations, which is obtained from the trained model available in Stanford Core NLP package of Java.
- When we checked through the input articles, we found a unique set of important texts occurring in a few places. These texts were in table format. The sentence tokenization of the NLTK package does not recognize each row of table as single sentence. Instead it considers the entire table 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 tables which considers each row as a separate sentence and wrote a certain set of rules to parse the row data.

Pending Issues:

- Accuracy can be improved by increasing the set of rules required to cover a wider range of extraction in WORK template.
- In few cases, Spacy's Named Entity Recognition fails to recognize properly whether a word is a PERSON or an ORG. Example. EURO is an organization's name but NER recognizes it as a PERSON.
- Required to annotate data and use Deep Learning techniques to extract more accurate information.

Potential Improvements:

- Training the spacy model for the given articles with tagged entities could result in improving the recognizer's accuracy.
- Training the custom named entity for POSITION with a large golden corpus could result in better detection of the job titles.