COMP 6751 Natural Language Analysis

Project 2 Report 1 Pipeline

Student: Yixuan Li   40079830

Table of Contents

Expectations of originality:

I, student 40079830, certify that this submission is my original work and meets the Faculty's Expectations of Originality.

Date: October 27, 2020

I. Modules documentation

1) Module 1: Sentence Splitting

```
sents = nltk.sent_tokenize(raw)
```

nltk.sent_tokenize returns a sentence-tokenized copy of text using sentence tokenizer *PunktSentenceTokenizer*.

2) Module 2: Word Tokenization

For each sentence, first we get a list of word tokens by calling

```
words = nltk.word_tokenize(sent)
```

nltk.word_tokenize tokenizes a string and split off punctuations and words and returns a list of word tokens.

3) Module 3: Named Entity Module

After word tokenization on each sentence, I pass the list of word tokens to a named entity module. In the named entity module, I leverage the NLTK's recommended named entity chunker to detect the word tokens which belong to the same named entity.

```
ne_parse_tree = nltk.ne_chunk(words)
```

nltk.ne_chunk tags *[Person, Organization, Location]* and returns a parse tree where the tokens belonging to the same named entity are under the same subtree.

Then I traverse the parse tree and merge the word tokens under the same subtree and return the merged list of word tokens ***word_list_merged*** and a set of name entities ***name_entities***.

(I enhanced this module by using external library *nltk.tag.stanford.StanfordNERTagger*, and I will discuss the improvement and compare the results in Improvement section below.)

4) Module 3: POS Tagging

After the named entity module, I directly pass ***word_list_merged*** to my POS module and call

```
pos_tags = nltk.pos_tag(word_list_merged)
```

nltk.pos_tag tags the given list of tokens.

(I enhanced this module by using external library *nltk.tag.stanford.StanfordPOSTagger*, and I will discuss the improvement and compare the results in Improvement section below.)

5) Module 4: Earley parser and Feature-based context-free grammars

I used the feature-based Earley Chart Parser and wrote the grammar in “grammar.fcfg” file.

Here is the grammar productions:

```
% start S
# ###################
# Grammar Productions
# ###################

# S expansion productions
S -> NP[NUM=?n] VP[NUM=?n] | CC NP[NUM=?n] VP[NUM=?n] | NP[NUM=?n] VP[NUM=?n] DATE
S -> NP[NUM=?n] VP[NUM=?n] CC TV[NUM=?n] INTRO S
S -> NP[NUM=?n] TV[NUM=?n] INTRO S | NP[NUM=?n] TV[NUM=?n] INTRO S CC IN S
S -> PP COMMA NP[NUM=?n] VP[NUM=?n] | DATE COMMA NP[NUM=?n] VP[NUM=?n] | DATE
COMMA PP COMMA NP[NUM=?n] VP[NUM=?n]
S -> PP COMMA NP[NUM=?n] TV[NUM=?n] INTRO S | DATE COMMA NP[NUM=?n] TV[NUM=?n]
INTRO S
S -> PP COMMA DATE COMMA NP[NUM=?n] TV[NUM=?n] NP[NUM=?n] INTRO S
S -> RB COMMA NP[NUM=?n] VP[NUM=?n]

### -------------------------------------------------------- ###
# NP expansion productions
NP[NUM=?n] -> DT[NUM=?n] Nom[NUM=?n] | DT[NUM=?n] Nom[NUM=?n] PP | Nom[NUM=?n] PP
NP[NUM=?n] -> PRP | Nom[NUM=?n]
NP[NUM=?n] -> PRP NP[NUM=?n] | PRP DATE PP
NP[NUM=?n] -> DT[NUM=?n] JJ NP[NUM=?n]
NP[NUM=?n] -> N[NUM=?n] CC NP[NUM=?n] | NNP[NUM=?n] CC NP[NUM=?n]
Nom[NUM=?n] -> N Nom[NUM=?n] | N | NNP[NUM=?n]

### -------------------------------------------------------- ###
# VP expansion productions
VP[TENSE=?t, NUM=?n] -> VP[TENSE=?t, NUM=?n] PP
VP[TENSE=?t, NUM=?n] -> TV[TENSE=?t, NUM=?n] NP | RB TV[TENSE=?t, NUM=?n] NP |
MD[TENSE=?t] TV[TENSE=inf] NP RB
VP[TENSE=?t, NUM=?n] -> MD[TENSE=?t] TV[TENSE=inf] NP | MD[TENSE=?t]
AUX[TENSE=inf] JJ | AUX[TENSE=?t] JJ | MD[TENSE=?t] AUX[TENSE=inf] | MD[TENSE=?t]
IV[TENSE=inf]
VP[TENSE=?t, NUM=?n] -> IV[TENSE=?t] TO VP[TENSE=inf]
VP[TENSE=?t, NUM=?n] -> IV[TENSE=?t] PP

### -------------------------------------------------------- ###
# PP expansion productions
PP -> IN NP | IN DATE


# ###################
# Lexical Productions
# ###################

(omitted)
...
```

(Here, I omit the lexical productions in “grammar.fcfg” for saving spaces.)

Changes on Grammars compared to the CFG grammar:

- NUM, TENSE features

  Nouns have the property of being singular or plural, correspondingly, verbs have the property of being first, second, third person agreement. So I use the NUM and TENSE features. Similarly on determinants DT, auxiliaries AUX and model MD.
- Precisely specify the verb subcategories: TV (transitive verb), IV (intransitive verb)
- Integrate Date grammars in project 1

Possible improvement:

- Verb subcategories

  It is possible to more precisely specify the verbs into 4 sub-categories (intransitive verb, transitive verb, dative verb and sentential verb).

6) Improvement section: Parser enhancement and comparison with base version

In the submission, I designed another similar pipeline in "main_stan.py" file and I use StanfordPOSTagger and StanfordNERTagger in Part-of-Speech Module and Named Entity Module, and I am going to show comparative results on them.

a) Comparison between "nltk.pos_tag" and "StanfordPOSTagger"

Input	"nltk.pos_tag" POS result in "main_base.py"	StanfordPOSTagger POS result in "main_stan.py"
Input file: "data/compare1.txt" **Sentence:** O'Malley intended to share it with her on Tuesday at his desk and anticipated that the crunchy treat would delight them both.	[[("O'Malley", 'NNP'), ('intended', 'VBD'), ('to', 'TO'), ('share', 'NN'), ('it', 'PRP'), ('with', 'IN'), ('her', 'PRP'), ('on', 'IN'), ('Tuesday', 'NNP'), ('at', 'IN'), ('his', 'PRP$'), ('desk', 'NN'), ('and', 'CC'), ('anticipated', 'VBD'), ('that', 'IN'), ('the', 'DT'), ('crunchy', 'NN'), ('treat', 'NN'), ('would', 'MD'), ('delight', 'VB'), ('them', 'PRP'), ('both', 'DT')]]	[[("O'Malley", 'NNP'), ('intended', 'VBD'), ('to', 'TO'), ('share', 'VB'), ('it', 'PRP'), ('with', 'IN'), ('her', 'PRP'), ('on', 'IN'), ('Tuesday', 'NNP'), ('at', 'IN'), ('his', 'PRP$'), ('desk', 'NN'), ('and', 'CC'), ('anticipated', 'VBD'), ('that', 'IN'), ('the', 'DT'), ('crunchy', 'JJ'), ('treat', 'NN'), ('would', 'MD'), ('delight', 'VB'), ('them', 'PRP'), ('both', 'DT')]]

As we can see that, in "main_base.py", nltk.pos_tag tags "share" as a Noun, whereas in "main_stan.py" StanfordPOSTagger tags "share" as a Verb which is more proper here. And nltk.pos_tag also improperly tags "crunchy" as a Noun, but StanfordPOSTagger correctly tags it as an Adjective.

b) Comparison between “nltk.ne_chunk” and “StanfordNERTagger”

Input	“nltk.ne_chunk” Named Entity result in “main_base.py”	StanfordNERTagger Named Entity result in “main_stan.py”
Input file: “data/compare1.txt” **Sentence:** **O'Malley** intended to share it with her on Tuesday at his desk and anticipated that the crunchy treat would delight them both.	Name entities: set()	Name entities: {"O'Malley"}
Input file: “data/compare2.txt” **Sentence:** On Monday, September 17, 2018, **John O'Malley** promised his colleague **Mary** that he would put a replacement apple in the office fridge.	Name entities: {"John O'Malley"}	Name entities: {"John O'Malley", 'Mary'}

We can compare the result between nltk.ne_chunk and StanfordNERTagger. In most cases, StanfordNERTagger can detect more precisely than vanilla nltk.ne_chunk function.

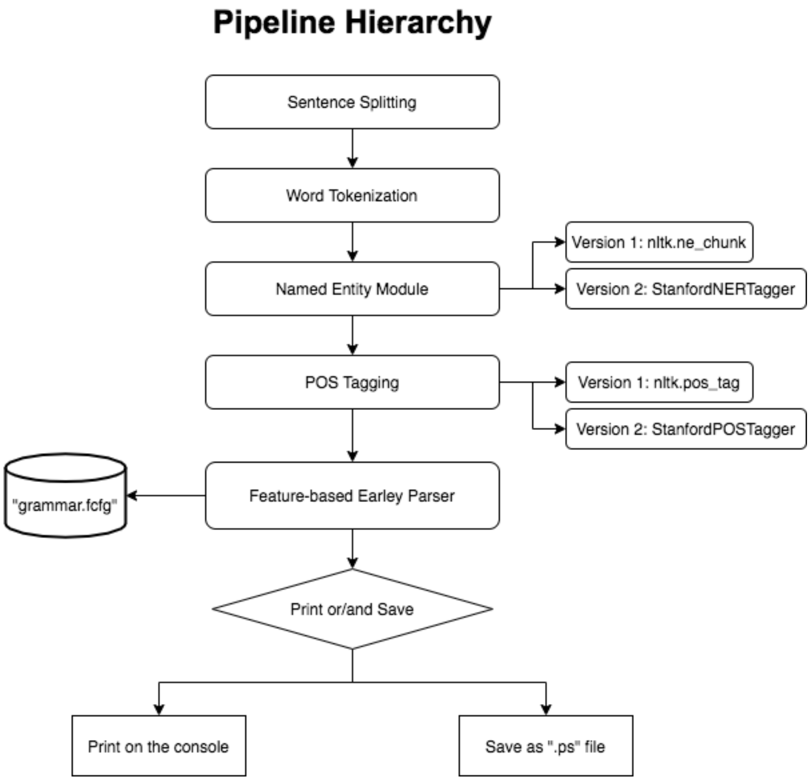
However, StanfordNERTagger does not always perform better than nltk.ne_chunk. So I use the union result of named entity module of both “main_base.py” and “main_stan.py” for the sentence example 9 in my Demo report.

Input	“nltk.ne_chunk” Named Entity result	StanfordNERTagger Named Entity result
Input file: “data/sent9.txt” **Sentence:** **Sue** said that on Monday, September 17, 2018, **John O'Malley** promised his colleague **Mary** that he would put a replacement apple in the office fridge and that **O'Malley** intended	Name entities: {"O'Malley", 'Sue', "John O'Malley"}	Name entities: {"John O'Malley", 'Mary', "O'Malley"}

to share it with her on Tuesday at his desk.		

We can see that StanfordNERTagger doesn't return "Sue" as a named entity, and at the meanwhile, nltk.ne_chunk doesn't return "Mary" as a named entity. But their union results include all the named entities in the sentence.

II. Pipeline



III. Limitations

PP attachment ambiguity is the limitation on feature-based Earley Parser grammars.

In my Demo report, grammars on sentence 2, 3, 4, 5 generate 2 possible parse trees due to the PP constituents (e.g. in the fridge, in his office). And it may generate more possible trees if the sentence is long and contains multiple PP constituents, such as sentence 9 (*Sue said that **on Monday**, September 17, 2018, John O'Malley promised his colleague Mary that he would put a replacement apple **in the office fridge** and that O'Malley intended to share it **with her** **on Tuesday** **at his desk**.*)

However, not every PP constituent has ambiguity. For example, in the example above "on Monday" doesn't have such ambiguity since there is no Noun Phrase or Verb Phrase before it in the sentence or clause which it belongs to.

In other example such as sentence 4 (*On Monday, John ate the apple* ***<u>in his office</u>****.*), "in his office" could possibly attach to the verb "ate" or the noun phrase "the apple". So my grammar generates 2 parse trees as well.

IV. References

1. References in Project 1 Report and Project 1 delivery
2. NLTK book chapter 7,8,9 (http://www.nltk.org/book/)
3. NLTK API chunk package (https://www.nltk.org/api/nltk.chunk.html)
4. StanfordPOSTagger (https://nlp.stanford.edu/software/tagger.html)
5. StanfordNERTagger (https://nlp.stanford.edu/software/CRF-NER.html)
6. StanfordNERTagger code example (https://pythonprogramming.net/named-entity-recognition-stanford-ner-tagger/)
7. Save NLTK drawn parse tree to image file (https://stackoverflow.com/questions/23429117/saving-nltk-drawn-parse-tree-to-image-file)

V. Appendix

The parse tree diagrams are saved as a ".ps" files under the directory "results/".
To view the parse trees:

- If you are on Windows, it is necessary to install **ImageMagick** and use the command below

  ```
  $ convert "output1.ps" "tree1.png"
  ```

  to convert the ".ps" file to an image file.
- If you are on Mac, you can open the ".ps" file using the application *Preview*, and it will automatically open it as a pdf file.

For convenience, I have converted already and saved all the diagrams in the directory **"code/got_results",** and the directory name corresponds to the files listed in the "code/data", and the diagrams are also used in my Demo report.