REPORT

PROJECT 2

PREPARED FOR

Dr. Sabine Bergler

PREPARED BY

Karandeep

40104845

TABLE OF CONTENTS

Introduction:

About NLTK - NLTK aka the Natural Language Toolkit, is a suite of open source Python modules, data sets, and tutorials supporting research and development in Natural Language Processing. It contains text processing libraries for tokenization, parsing, classification, stemming, tagging and semantic reasoning. It also includes graphical demonstrations and sample data sets as well as accompanied by a cook book and a book which explains the principles behind the underlying language processing tasks that NLTK supports.

Purpose:

Hidden information often lies deep within the boundaries of what we can perceive with our eyes and our ears. Some look to data for that purpose, and most of the time, data can tell us more than we thought was imaginable. But sometimes data might not be clear cut enough to perform any sort of analytics. Language, tone, and sentence structure can explain a lot about how people are feeling, and can even be used to predict how people might feel about similar topics using a combination of the Natural Language Toolkit, a Python library used for analyzing text, and machine learning.

For this project I was to develop a pipeline and a **CFG** to parse given sentences. The console has options to run **prepipeline** either on an inbuilt data or takes sentence from console to parse. The pipeline *Preprocess* parses by taking text using the NLTK corpus access commands. Analysis of data has been done in a way that is easily comprehensible. Overall goal of the project was to create a pipeline and a CFG which filters data by using NLTK access commands and creates sentence trees and visualize them in a more informative way. NLTK is not perfect and lacks in some areas which is being explored below.

Getting Started:

1. Open CMD (for Windows), Terminal (for MacOS).
2. Goto directory of the script.
3. Type in command “python3 preprocess.py”

CFG (Context Free Grammar):

English Grammar CFG:

Explanation:

1. Module Name - **sentence_parser**()
2. It parses sentences into a parse tree that is a representation of syntactic structure, on which semantic interpretation can be based.
3. It takes raw sentences either from inbuilt data or from the console.
4. It uses pipelines sent_tokenizer and word tokenizer to split sentences into words and then draws sentences into a parse tree of the form **NP VP POS** using the **PrettyPrint**.
5. This Context free grammar that covers as much different text, it covers the validation data provided below.

Validation Data :

1. John ate an apple.
2. John ate the apple at the table.
3. On Monday, John ate the apple in the fridge.
4. On Monday, John ate the apple in his office.
5. On Monday, John ate refrigerator apple in his office.
6. Last week, on Monday, John finally took the apple from the fridge to his office.
7. Last Monday, John promised that he will put an apple in the fridge. He will eat it on Tuesday at his desk. It will be crunchy.
8. On Monday, September 17, 2018, John O'Malley promised his colleague Mary that he would put a re- placement apple in the office fridge. O'Malley intended to share it with her on Tuesday at his desk and anticipated that the crunchy treat would delight them both. But she was sick that day.
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.

Issues:

Although the CFG parses above data perfectly but is limited to provided context, in order to make it a generalized CFG more words can be added over time to improve the coverage.

OUTPUT:

A parse tree of the form,

S (NP (NNP John)) (VP (VBD ate) (NP (DT an) (NN apple))) (POS .))(

Earley Parser:

Explanation:

1. Earley Parser has been used to parse the CFG grammar. EarleyChartParser from NLTK uses an algorithm to parse sentences using the provided grammar using the **PrettyPrint** to print the sentences in the form trees.
2. The context free grammar which has been manually written uses the limited dictionary of words to find patterns from the word list and Parser then makes a tree out of the list.

Issues:

EarleyChartParser being the native NLTK tool parses almost perfectly according to the grammar provided.

OUTPUT:

Output is a pretty printed tree with nodes as children. It is of type NLTK.TREE.tree.

S (NP (NNP John)) (VP (VBD ate) (NP (DT an) (NN apple))) (POS .))

Pipeline:

Tokenization:

Explanation:

1. Module Name - **tokenizer**()
   - This module is using a straightforward approach of firstly reading the data from the source file. And splitting the sentences using NLTK's **sent_tokenize**().
2. Most samples used for demo have line breaks ("**\n**")**.** So the tokenizer() splits the sentences on line breaks and is accurate most of the time in splitting data into sentences.
3. Implemented tokenizer that uses unsupervised learning algorithm and is itself pre trained to identify where text should split.

Issues:

- Splitting by looking for line breaks only works when there is a line break encountered. So in order to overcome this sent_tokenize() was used to split on appropriate sentence breaks.
- Sent_tokenize can't detect if to split when encountered 2 two dots (.) for abbreviations, for e.g.- Er. , when encountered this sent_tokenize() splits on the end of Er. and the name (Noun) is moved to the next line.

Enhancements:

- By using regex library and suitable pattern which is “r"(?<= [.(a-zA-z]{3})\.(?!=(\n))"” and “r"(?<= [a-zA-z]{2})\.(?!=(\n))"”, I was able to solve the abbreviation problem and sent_tokenize did not split on the end of Er. or any other abbreviations, like, Dr. Sr. Mr. MRS.

OUTPUT:

- Output of this module is the list of strings of words. E.g.- [“I” , “love”, “Montreal”].
- The output is given after passing through the word tokenizing part of the module explained below.

Sentence Splitter:

Explanation:

1. Module Name - **tokenizer**()
    - Splitting the words that are the part of the module tokenizer.
    - NTLK’s word_tokenize() is quite accurate for splitting the words on point without any false positives.
2. Most samples used for demo have line breaks (“**\n**”)**.** So the tokenizer() splits the sentences on line breaks and is accurate most of the time in splitting data into sentences. For the words word_tokenize(), the NLTK’s own method is smart enough to split on appropriate word ends.
3. With the help of earlier enhancement of sentence splitter, the word_tokenize() was able to split words form words on correct ends.
4. word_tokenize() assumes that the data has been splitted into sentences.

Issues:

- By default split function separates using whitespace in sentences, so it cannot separate, for example, Karan’s iphone to [‘Karan’, ‘’s’, ‘iphone’], instead it separates to [‘Karan’s’, ‘iphone’].

Enhancements:

- The data set of *Reuters* by NLTK had error in recognising the ‘<’ sign from the raw data. In order to overcome this, I went through a list of identified the token of words and when found a similar pattern of word I replaced the random characters placed there with ‘<’.

OUTPUT:

- Output of this module is the list of strings of words. E.g.- [“I” , “love”, “Concordia”, “.”].

POS Tagger:

Explanation:

1. Module Name - **pos_tagger**()
    - Identifying words is the most tedious part of text parsing. And NLTK's pos_tag() very easily recognises the words and there tags and outputs the findings in a list of tuples.
    - One thing to note here is that pos_tag() tags all numbers/integers as "CD" - Cardinal digit. Which is challenging as it becomes difficult to recognise which is the date and which is the measured entity.
2. pos_tag() uses their CFG to identify, which is a library of hundreds of smart regex patterns to recognise the type of word.
3. pos_tagger() takes a list of words as input and loops through the list and tags the words by adding them in a tuple along with the tag name and then finally appending to the master list of tuples.
4. pos_tagger() assumes that the data has been splitted into words and is a list.

Issues:

- pos_tag() tags all numbers/integers as "CD" - Cardinal digit. Which is challenging as it becomes difficult to recognise which is the date and which is the measured entity. For e.g.: it recognises "2.3 billion" as [("2.3", "CD"),("billion", "billion")].
- All data had one mistake as it had placed 'lt&;'in the place '<'. And due to this extra characters were added to the list of tuples of words with tags.

Enhancements:

- To correct the correct form/ type of tag to misplaced word, I went through the list of words and when a pattern of **"NN: {<CC><NN><:>}"** was found it would replace the item in the list with ('<', 'NN').

OUTPUT:

- Output of this module is the list of tuples with word as position 0 of tuple and tag at position2 of tuple.

Measured Entity Detection: **UPDATED**

Explanation:

1. Module Name - **measuredEntityDetection**()
    - measuredEntityDetection() module's goal is to identify the pattern where numbers and proper nouns come together. Because pos_tagger() does not mark the numbers explicitly, measureEntityDetection module does this job.
2. It takes a list of tuples as input from the pos tagger mainly, and then uses **RegexpParser**() to match the pattern of this format - **'ME: {<CD><NNS>}'** in this ME is the tag of the word.

3. After catching the word it uses **Unit Gazetter** to identify the NNS if it is in the list if not then it ignores and moves to the next word. But if matches and then does the chunking.
4. Measured entity clubs the words together which it seems suitable according to the pattern.
5. measuredEnityDetection() assumes that the data has been splitted into tuples with words and tags and is a list.

Issues:

- Because of the scope of the english language, the script cannot be scaled to millions of word sets, but in most of the data set measured entities were able to identify the entity and mark them in a tree. But the Unit gazetteer is limited to the dictionary and according to architecture it ignores the words if not in the dictionary.

OUTPUT:

- Output of this module is the list of tuples with word as position 0 of tuple and tag at position 2 of tuple and where there is tag applied then is makes a tree of of type - "tree.NLTK.TREE" type, e.g. **- (ME 51,000/CD tonnes /NNS)**

Date Recognition CFG:

Explanation:

1. Module Name - **dateRecognizer**()
2. dateRecoginzer() module's goal is to identify the pattern where numbers and proper nouns club together.
3. Technique used is dateRecognizer() takes a list of tuples from the enityRecognition() and it loops through every tuple which has either a cardinal digit or a noun from months list.
4. After getting the favourable word I check in a radius of 4 words of both, and check if the word is a preposition and a proper noun.
5. Afterrecoginsing the date it appends the string of date to list and passes to dateParser().
6. It checks the following format:
    - First of November 2018
    - First of November
    - 1 of November of year 2018
    - 1st of November of year 2018
    - 1 of November
    - 1s of November
    - November 10 2018
    - November 10th 2018
    - November 1
    - Nov 1
    - November 1st
    - 1 November 2018

- 1st November 2018
- 1 November
- 1st November
- November 2018
- 2018
- 01-11-2018
- 01/11/2018
- 01.011.2018
- 01 01 2018

Issues:

- Date recogniser can not recogniser if pattern of matching is out of scope of the pattern on which the date is decided. For e.g. - “Frank the Second” will be recognised as a date, where else it clearly is a name of a person.

OUTPUT:

- A list of date strings which one by one is given to dateparser to parse date out of string.

Date Parser CFG:

Explanation:

1. Module Name - **dateParser**()
2. dateParser() module’s goal is to identify the pattern by matching the grammar specified in NLTK.CFG.
3. Technique used is dateParser() takes a date and matches the string by splitting and feeding into the tree.
4. dateParser() after recognising the date makesa NLTK.tree which can be viewed as a normal tree with child and root.
5. dateParser() prints the dates in the format matched with pattern.
6. It can parse the following format:
    - First of November 2018
    - First of November
    - 1 of November of year 2018
    - 1st of November of year 2018
    - 1 of November
    - 1s of November
    - November 10 2018
    - November 10th 2018
    - November 1
    - Nov 1
    - November 1st
    - 1 November 2018
    - 1st November 2018
    - 1 November

- 1st November
- November 2018
- 2018
- 01-11-2018
- 01/11/2018
- 01.11.2018
- 01 01 2018

Issues:

- dateParser() prints all the matches of string which match in grammar. For eg. if: “2018”, it will give following:

```
(DATE (YEAR (NUM 2) (NUM 0) (NUM 1) (NUM 8)))
(DATE (MONTH (NUM 2) (NUM 0)) (DAY (NUMDAY (NUM 1) (NUM 8))))
(DATE (DAY (NUMDAY (NUM 2) (NUM 0))) (MONTH (NUM 1) (NUM 8)))
```

Now, first one is right as it parsed 2018 as year, but as Grammar has to match some dates 02 23 or 31 01 where these are dates, in which case above would have been right.

OUTPUT:

- A tree with DATE as root and DAY, MONTH, CHR, INP and YEAR as child leaves. For e.g. for - date December 31 it will give **- (DATE (MONTH December) (DAY (NUMDAY (NUM 3) (NUM 1))))**

Named Entity Detection:

Explanation:

6. Module Name - **named_entity**()
7. This module’s goal is to identify names of persons, organizations etc. from the pos tagged list given by measured Entity module.
8. Module identifies the names and adds appropriate tags with the words catched.
9. It takes a list of tuples, inside the list are tuples in which 0th place has word and 1st place has the word.
10. It replaces the identified words and adds it into a NLTK tree object and places it in the list.
11. It returns a list of names identified.

Issues:

The only issue with this module is sometimes if the name of a person is more than one it identifies it as a separate word. So for example if there is a sentence called “ John O’Malley ate an apple from the fridge.”

Now this is a single name but named_entity() gives : [“John”, “O”, “Malley”], it gives the whole name in parts.

This can be fixed with work arounds, but that will be out of scope of this project.

OUTPUT:

A list of strings identified as names.

References:

1. https://www.nltk.org/_modules/nltk/tokenize/treebank.html#TreebankWordTokenizer
2. https://www.nltk.org/_modules/nltk/tag.html#pos_tag
3. https://nlp.stanford.edu/software/pos-tagger-faq.html#tagset
4. http://www.ibiblio.org/units/measurements.html
5. http://www.ibiblio.org/units/index.html
6. http://www.nltk.org/book/