



REPORT PROJECT 1

PREPARED FOR

Dr. Sabine Bergler

PREPARED BY

Karandeep

40104845

TABLE OF CONTENTS

Introduction:	2
Purpose:	3
Pipeline:	3
Getting Started:	3
Tokenization:	3
Explanation:	3
Issues:	4
Enhancements:	4
OUTPUT:	4
Sentence Splitter:	4
Explanation:	4
Issues:	5
Enhancements:	5
OUTPUT:	5
POS Tagger:	5
Explanation:	5
Issues:	5
Enhancements:	6
OUTPUT:	6
Measured Entity Detection:	6
Explanation:	6
Issues:	6
OUTPUT:	6
Date Recognition CFG:	6
Explanation:	6
Issues:	7
OUTPUT:	7
Date Parser CFG:	7
Explanation:	7
Issues:	8
OUTPUT:	8
References:	9

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:

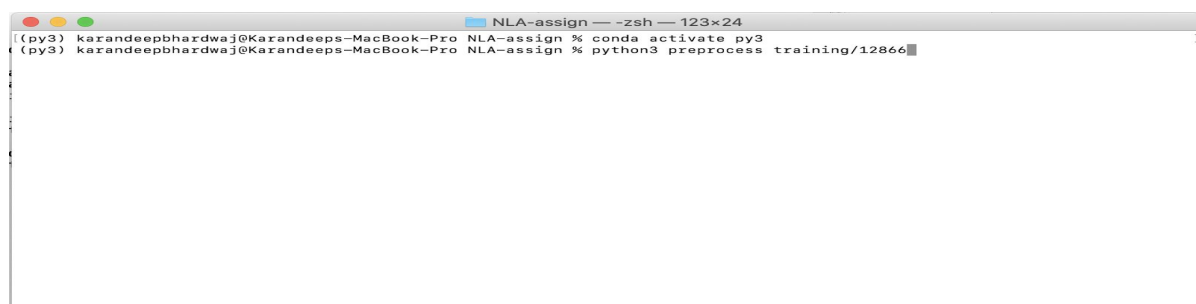
Hidden information often lies deep within the boundaries of what we can perceive with our eyes and our ears. Some look to data for that purpose, and most of the time, data can tell us more than we thought was imaginable. But sometimes data might not be clear cut enough to perform any sort of analytics. 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. 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 which filters data by using NLTK access commands and visualize them in a more informative way. NLTK is not perfect and lacks in some areas which is being explored below.

Pipeline:

Getting Started:

1. Open CMD (for Windows), Terminal (for MacOS).
2. Goto directory of the script.
3. Type in command "python3 preprocess.py training/12866"

A screenshot of a terminal window on a macOS system. The window title is "NLA-assign - zsh - 123x24". The prompt is "[(py3) karandeepbhardwaj@Karandeeps-MacBook-Pro NLA-assign %". The command entered is "conda activate py3" followed by "python3 preprocess training/12866". The cursor is at the end of the second command.

```
[(py3) karandeepbhardwaj@Karandeeps-MacBook-Pro NLA-assign % conda activate py3
(py3) karandeepbhardwaj@Karandeeps-MacBook-Pro NLA-assign % python3 preprocess training/12866
```

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.
 - NLTK'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 from 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:

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. Measured entity clubs the words together which it seems suitable according to the pattern.
4. measuredEntityDetection() 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 was able to identify the entity and mark them in a tree. But if a similar pattern comes in a sentence which matches with the pattern of chunk then it can falsely be taken as measured Entity.

OUTPUT:

- Output of this module is the list of tuples with word as position 0 of tuple and tag at position2 of tuple and where there is tag applied then it makes a tree of type - "tree.NLTK.TREE" type, e.g. - (**ME 51,000/CD subscribers/NNS**)

Date Recognition CFG:

Explanation:

1. Module Name - **dateRecognizer()**

2. dateRecognizer() 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 entityRecognition() 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. After recognising 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 recognise 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 makes a 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))))

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/>