NLP LAB – 8

Anveshak Rathore
181060012
Final Year B.Tech Electronics

AIM: To perform Stochastic Tagging in NLP.

SOFTWARE: Python 3.8.6, Visual Studio Code

THEORY:

Tagging is a kind of classification that may be defined as the automatic assignment of description to the tokens. Here the descriptor is called tag, which may represent one of the part-of-speech, semantic information and so on.
Now, if we talk about Part-of-Speech (POS) tagging, then it may be defined as the process of assigning one of the parts of speech to the given word. It is generally called POS tagging. In simple words, we can say that POS tagging is a task of labelling each word in a sentence with its appropriate part of speech. We already know that parts of speech include nouns, verb, adverbs, adjectives, pronouns, conjunction, and their sub-categories.
Most of the POS tagging falls under Rule Base POS tagging, Stochastic POS tagging, and Transformation based tagging.

Stochastic POS Tagging

Another technique of tagging is Stochastic POS Tagging. Now, the question that arises here is which model can be stochastic. The model that includes frequency or probability (statistics) can be called stochastic. Any number of different approaches to the problem of part-of-speech tagging can be referred to as stochastic tagger.
The term 'stochastic tagger' can refer to any number of different approaches to the problem of POS tagging. Any model which somehow incorporates frequency or probability may be properly labelled stochastic.
The simplest stochastic taggers disambiguate words based solely on the probability that a word occurs with a particular tag. In other words, the tag encountered most frequently in the training set with the word is the one assigned to an ambiguous instance of that word. The problem with this approach is that while it may yield a valid tag for a given word, it can also yield inadmissible sequences of tags.
The simplest stochastic tagger applies the following approaches for POS tagging –

Word Frequency Approach:

In this approach, the stochastic taggers disambiguate the words based on the probability that a word occurs with a particular tag. We can also say that the tag encountered most frequently with the word in the training set is the one assigned to an ambiguous instance of that word. The main issue with this approach is that it may yield inadmissible sequence of tags.

Tag Sequence Probabilities:

It is another approach of stochastic tagging, where the tagger calculates the probability of a given sequence of tags occurring. It is also called n-gram approach. It is called so because the best tag for a given word is determined by the probability at which it occurs with the n previous tags.

Properties of Stochastic POS Tagging:

Stochastic POS taggers possess the following properties −

- This POS tagging is based on the probability of tag occurring.
- It requires training corpus
- There would be no probability for the words that do not exist in the corpus.
- It uses different testing corpus (other than training corpus).
- It is the simplest POS tagging because it chooses most frequent tags associated with a word in training corpus.

PROGRAM:

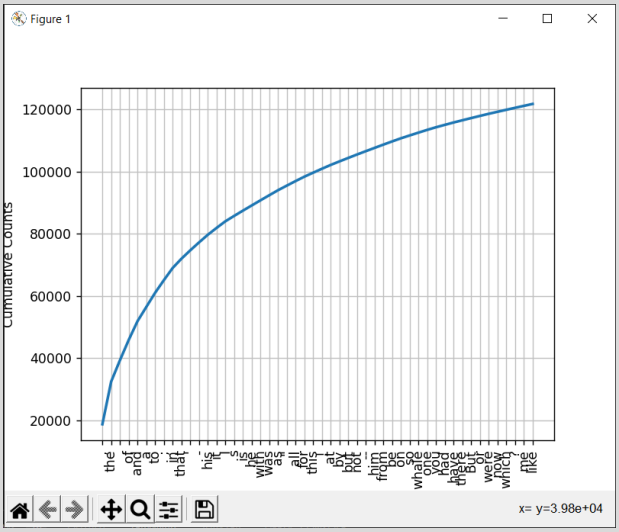
```
from collections import Counter
text = "It is necessary for any Data Scientist to understand Natural Language Processing"
text = text.lower()
the_count = Counter(text)
print(the_count)

import nltk
from collections import Counter
text = "It is necessary for any Data Scientist to understand Natural Language Processing"
text = text.lower()
tokens = nltk.word_tokenize(text)
pos = nltk.pos_tag(tokens)
the_count = Counter(tag for _, tag in pos)
print(the_count)


from nltk.corpus import gutenberg
text = gutenberg.words('melville-moby_dick.txt')
print(len(text))
print(text[:100])

fdistribution = nltk.FreqDist(text)

fdistribution.plot(50, cumulative=True)


text = "Anna Karenina by Leo Tolstoy. So, we've to learn History, French, ,Germann and the
like! Yay!"

#Sentence tokenization
from nltk.tokenize import sent_tokenize
sentences = sent_tokenize(text)
print(len(sentences), 'sentences:', sentences)

#Word tokenization
from nltk.tokenize import word_tokenize
words = word_tokenize(text)
print(len(words), 'words:', words)


#Filter out stop words
from nltk.corpus import stopwords
stop_words = stopwords.words('english')
print(len(stop_words), "stopwords:", stop_words)
words = [word for word in words if word not in stop_words]
print(len(words), "without stopwords:", words)
```

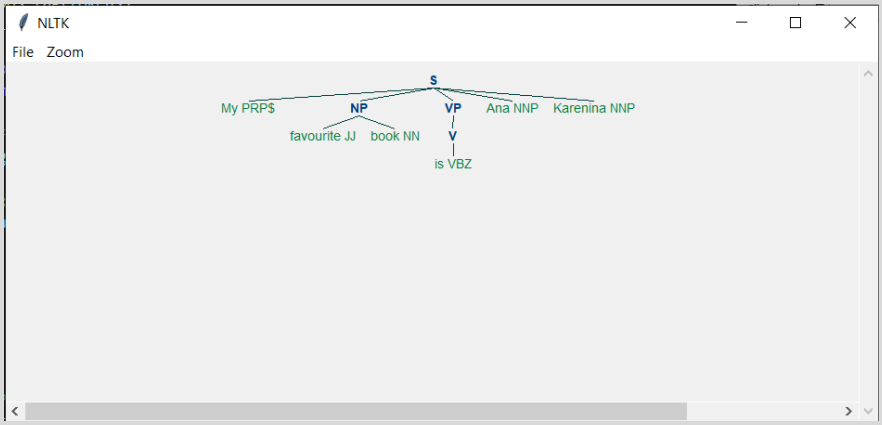
```
#Tokenize the text first, then filter out the stopwords:
text = "Anna Karenina by Leo Tolstoy. So, we've to learn History, French, ,Germann and the
like! Yay!"

from nltk.tokenize import word_tokenize
words = word_tokenize(text)

print(len(words), "in original text:", words)

words = [word for word in words if word not in stop_words]
print(len(words), "without stopwords:", words)

import string
punctuations = list(string.punctuation)
print(punctuations)
words = [word for word in words if word not in punctuations]
print(len(words), "words without stopwords and punctuations:", words)

from nltk import word_tokenize, pos_tag
text = "My favourite book is Ana Karenina"
tokens = word_tokenize(text)


y=pos_tag(tokens)
print(pos_tag(tokens))

from nltk import pos_tag, word_tokenize, RegexpParser
sample_text = "My favourite book is Ana Karenina"

# Find all parts of speech in above sentence
tagged = pos_tag(word_tokenize(sample_text))

#Extract all parts of speech from any text
chunker = RegexpParser("""
                       NP: {<DT>?<JJ>*<NN>}    #To extract Noun Phrases
                       P: {<IN>}               #To extract Prepositions
                       V: {<V.*>}              #To extract Verbs
                       PP: {<p> <NP>}          #To extract Prepositional Phrases
                       VP: {<V> <NP|PP>*}      #To extract Verb Phrases
                       """)

# Print all parts of speech in above sentence
output = chunker.parse(tagged)
print("After Extracting\n", output)

output.draw()
```

OUTPUT:

```
Counter({' ': 11, 'a': 9, 's': 8, 'n': 8, 't': 7, 'e': 6, 'i': 5, 'r': 5, 'c': 3, 'o': 3,
'd': 3, 'u': 3, 'g': 3, 'y': 2, 'l': 2, 'f': 1, 'p': 1})


Counter({'NN': 3, 'JJ': 2, 'PRP': 1, 'VBZ': 1, 'IN': 1, 'DT': 1, 'NNS': 1, 'TO': 1, 'VB':
1})


260819


['[', 'Moby', 'Dick', 'by', 'Herman', 'Melville', '1851', ']', 'ETYMOLOGY', '.', '(',
'Supplied', 'by', 'a', 'Late', 'Consumptive', 'Usher', 'to', 'a', 'Grammar', 'School', ')',
'The', 'pale', 'Usher', '--', 'threadbare', 'in', 'coat', ',', 'heart', ',', 'body', ',',
'and', 'brain', ';', 'I', 'see', 'him', 'now', '.', 'He', 'was', 'ever', 'dusting', 'his',
'old', 'lexicons', 'and', 'grammars', ',',
'with', 'a', 'queer', 'handkerchief', ',', 'mockingly', 'embellished', 'with', 'all', 'the',
'gay', 'flags', 'of', 'all', 'the', 'known', 'nations', 'of', 'the', 'world', '.', 'He',
'loved', 'to', 'dust', 'his', 'old', 'grammars', ';', 'it', 'somehow', 'mildly', 'reminded',
'him', 'of', 'his', 'mortality', '.', '"', 'While', 'you', 'take', 'in', 'hand', 'to',
'school', 'others', ',']


3 sentences: ['Anna Karenina by Leo Tolstoy.', "So, we've to learn History, French, ,Germann
and the like!", 'Yay!']


24 words: ['Anna', 'Karenina', 'by', 'Leo', 'Tolstoy', '.', 'So', ',', 'we', "'ve", 'to',
'learn', 'History', ',', 'French', ',', ',', 'Germann', 'and', 'the', 'like', '!', 'Yay',
'!']


179 stopwords: ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you',
"you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he',
'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its',
'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom',
'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been',
'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the',
'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for',
'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',
'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under',
'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all',
'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not',
'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don',
"don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren',
"aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't",
'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',
"mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't",
'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]


19 without stopwords: ['Anna', 'Karenina', 'Leo', 'Tolstoy', '.', 'So', ',', "'ve", 'learn',
'History', ',', 'French', ',', ',', 'Germann', 'like', '!', 'Yay', '!']


24 in original text: ['Anna', 'Karenina', 'by', 'Leo', 'Tolstoy', '.', 'So', ',', 'we',
"'ve", 'to', 'learn', 'History', ',', 'French', ',', ',', 'Germann', 'and', 'the', 'like',
'!', 'Yay', '!']


19 without stopwords: ['Anna', 'Karenina', 'Leo', 'Tolstoy', '.', 'So', ',', "'ve", 'learn',
'History', ',', 'French', ',', ',', 'Germann', 'like', '!', 'Yay', '!']
```

```
['!', '"', '#', '$', '%', '&', "'", '(', ')', '*', '+', ',', '-', '.', '/', ':', ';', '<',
'=', '>', '?', '@', '[', '\\', ']', '^', '_', '`', '{', '|', '}', '~']


12 words without stopwords and punctuations: ['Anna', 'Karenina', 'Leo', 'Tolstoy', 'So',
"'ve", 'learn', 'History', 'French', 'Germann', 'like', 'Yay']


[('My', 'PRP$'), ('favourite', 'JJ'), ('book', 'NN'), ('is', 'VBZ'), ('Ana', 'NNP'),
('Karenina', 'NNP')]


After Extracting
 (S
  My/PRP$
  (NP favourite/JJ book/NN)
  (VP (V is/VBZ))
  Ana/NNP
  Karenina/NNP)
```





CONCLUSION:

We have successfully implemented Stochastic Tagging in Python.