AMA - Practical - QB - Solutions - By - Th3_

EXP 1: Text Corpus: Brown & Penn Treebank

AIM: Implement a program to use text corpus:

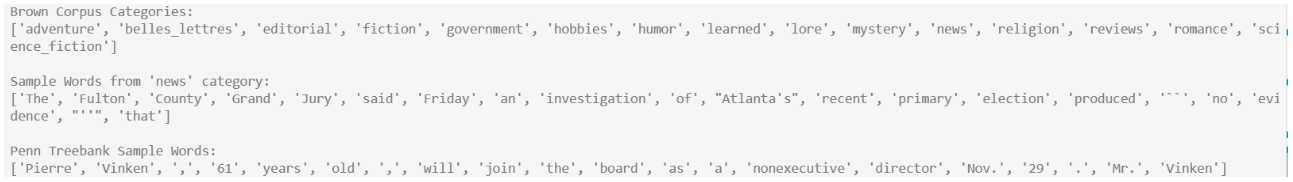
i) Brown Corpus OR ii) Penn Treebank Corpus

REQUIREMENTS(Software/Hardware):- Python Interpreter, Text Editor/IDE(VS Code), NLTK library

PROGRAM:

```
import nltk
from nltk.corpus import brown, treebank

nltk.download('brown')
nltk.download('treebank')

# Brown Corpus
print("Brown Corpus Categories:")
print(brown.categories())

print("\nSample Words from 'news' category:")
print(brown.words(categories='news')[:20])

# OR (you can skip this ↓)

# Penn Treebank Corpus
print("\nPenn Treebank Sample Words:")
print(treebank.words()[:20])
```

OUTPUT:



```
Brown Corpus Categories:
['adventure', 'belles_lettres', 'editorial', 'fiction', 'government', 'hobbies', 'humor', 'learned', 'lore', 'mystery', 'news', 'religion', 'reviews', 'romance', 'sci
ence_fiction']

Sample Words from 'news' category:
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', 'Friday', 'an', 'investigation', 'of', "Atlanta's", 'recent', 'primary', 'election', 'produced', '``', 'no', 'evi
dence', "''", 'that']

Penn Treebank Sample Words:
['Pierre', 'Vinken', ',', '61', 'years', 'old', ',', 'will', 'join', 'the', 'board', 'as', 'a', 'nonexecutive', 'director', 'Nov.', '29', '.', 'Mr.', 'Vinken']
```

CONCLUSION: The program successfully accessed and displayed sample data from the **Brown** and **Penn Treebank** corpora using NLTK.

QUESTION:

1. Define brown corpus. What's the main feature of brown corpus?

The Brown Corpus, compiled at Brown University in 1961, is one of the earliest and most widely used electronic corpora of American English. It contains approximately one million words of text from 500 samples, each roughly 2000 words long, drawn from 15 different categories of written prose published in the United States in 1961.

Its main feature is its balanced representation of different genres (e.g., news, editorial, religion, fiction, science fiction, humor, etc.), making it a valuable resource for studying the general characteristics of the English language and for tasks like frequency analysis and linguistic research.

2. What's the difference between brown corpus and penn treebank corpus?

The primary differences between the Brown Corpus and the Penn Treebank Corpus are:

Brown Corpus:

- **Annotation:** Primarily raw text, later POS-tagged.
- **Size/Source:** ~1M words, 15 genres, 1961.
- **Purpose:** General linguistic studies, frequency analysis, POS tagging.

Penn Treebank Corpus:

- **Annotation:** Richly annotated with syntactic (parse tree) and POS tags.
- **Size/Source:** ~4.5M words, mainly Wall Street Journal.
- **Purpose:** Syntactic parsing research.

EXP 2: Sentence & Word Segmentation

AIM: i) Write a program for Sentence Segmentation Techniques.

OR

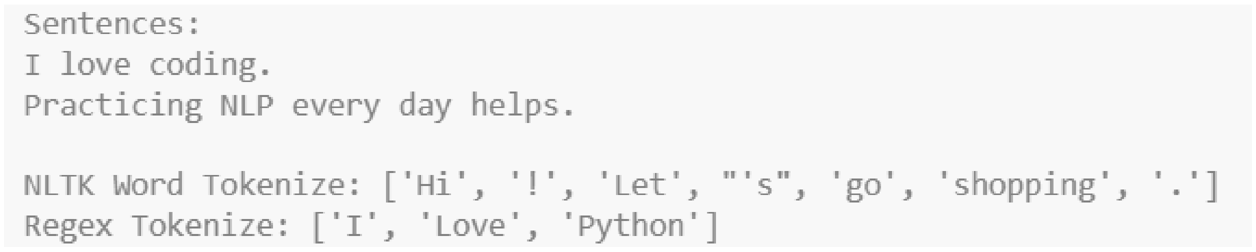
ii) Write a program Word Segmentation using Re and NLTK.

REQUIREMENTS(Software/Hardware):- Python, Text Editor/IDE(VS Code), NLTK, re library

PROGRAM:

```
# Sentence segmentation
import spacy
nlp = spacy.load("en_core_web_sm") # To install: python -m spacy download en_core_web_sm
doc = nlp("I love coding. Practicing NLP every day helps.")
print("Sentences:")
for sent in doc.sents:
    print(sent)

# OR (you can skip anyone  ↑ ↓)

# Word segmentation
from nltk.tokenize import word_tokenize, RegexpTokenizer
text = "Hi! Let's go shopping."
print("\nNLTK Word Tokenize:", word_tokenize(text))

tk = RegexpTokenizer(r'\s+', gaps=True)
print("Regex Tokenize:", tk.tokenize("I Love Python"))
```

OUTPUT:



```
Sentences:
I love coding.
Practicing NLP every day helps.

NLTK Word Tokenize: ['Hi', '!', 'Let', "'s", 'go', 'shopping', '.']
Regex Tokenize: ['I', 'Love', 'Python']
```

CONCLUSION: The program successfully implemented sentence segmentation using SpaCy and word segmentation using NLTK and Regular Expressions, demonstrating different tokenization techniques.

QUESTION

1. Split the text into word and sentence segmentation.
Text: "Hello Good Morning! Today is my NIP practical and I am very tensed

Segmentation of text:

Sentences: ["Hello Good Morning!", "Today is my NLP practical and I am very tensed"]
Words: ["Hello", "Good", "Morning", "!", "Today", "is", "my", "NLP", "practical", "and", "I", "am", "very", "tensed"]

This example demonstrates the initial NLP step of breaking text into sentences and words, crucial for further analysis. Sentence segmentation identifies complete thoughts, and word segmentation isolates individual words. Punctuation, like "!", is often treated as a separate word, depending on the NLP method.

2. State the library used for sentence segmentation and how to install that library.

SpaCy library is used for sentence segmentation, it provides reliable sentence segmentation and tokenization, which is crucial for accurate NLP processing.
To install SpaCy and its English language model:

1. **Install SpaCy**: pip install spacy
2. **Download the English language model** (encorewebsm):

```
python -m spacy download en_core_web_sm
```

EXP 3: Lemmatization & Stemming

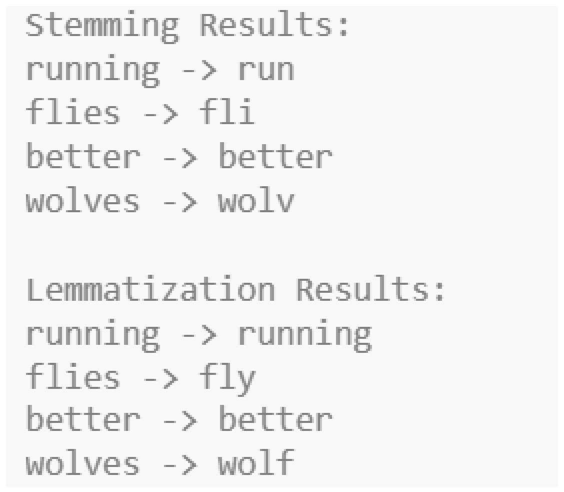
AIM: Apply various Lemmatization Techniques and Stemming Techniques such as porter stemmer OR Lancaster Stemmer OR Snowball Stemmer on text.

REQUIREMENTS(Software/Hardware):- Python Interpreter, Text Editor/IDE(VS Code), NLTK.

PROGRAM:

```
import nltk
from nltk.stem import PorterStemmer, WordNetLemmatizer

nltk.download('wordnet')

words = ["running", "flies", "better", "wolves"]
porter = PorterStemmer()
lemmatizer = WordNetLemmatizer()

print("Stemming Results:")
for w in words:
    print(w, "->", porter.stem(w))

print("\nLemmatization Results:")
for w in words:
    print(w, "->", lemmatizer.lemmatize(w))
```

OUTPUT:



```
Stemming Results:
running -> run
flies -> fli
better -> better
wolves -> wolv

Lemmatization Results:
running -> running
flies -> fly
better -> better
wolves -> wolf
```

CONCLUSION: The program successfully implemented stemming using the Porter Stemmer and lemmatization using the WordNet Lemmatizer, showing how both methods simplify words to their root forms.

QUESTION

1. What is the main difference between stemming and lemmatization in text processing?

The main difference between stemming and lemmatization lies in the output they produce and their approach to reducing words to their base form:

- **Stemming**: A more crude, heuristic process that chops off suffixes from words. It often results in a "stem" that is not necessarily a valid word (e.g., "running" -> "runn"). It's generally faster but less accurate.
- **Lemmatization**: A more sophisticated process that uses vocabulary and morphological analysis (knowledge of word structure) to reduce words to their dictionary or base form, known as a "lemma" (e.g., "running" -> "run"). The output is always a valid word. It's generally slower but more accurate.

2. Name one scenario where using a Porter Stemmer might be preferred over a Lancaster Stemmer.

The Porter Stemmer is a widely used and relatively gentle stemmer. A scenario where it might be preferred over a Lancaster Stemmer is when you need a less aggressive stemming that is more likely to preserve a recognizable (though not necessarily dictionary-valid) root, and when computational efficiency is a concern.

The Lancaster Stemmer is more aggressive, leading to shorter, less recognizable stems. The Porter Stemmer is preferred for a balance between reduction and form, avoiding over-stemming and information loss. For information retrieval, Porter is better for matching variations like "argue," "argued," and "arguing" without undesired truncation.

EXP 4: Text Normalization & N-Grams

AIM: a) Write program on Text Normalization using NLTK:

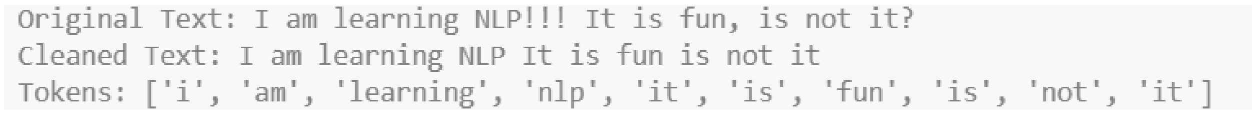
i)Tokenizing text ii)Removing special characters

REQUIREMENTS(Software/Hardware):- Python, Text Editor/IDE(VS Code), NLTK, re

PROGRAM:

```
import nltk, re, contractions
from nltk.tokenize import word_tokenize

nltk.download('punkt')

text = "I'm learning NLP!!! It's fun, isn't it?"
text = contractions.fix(text)                    # Expand contractions
clean_text = re.sub(r'[^a-zA-Z\s]', '', text)     # Remove special characters
tokens = word_tokenize(clean_text.lower())        # Tokenize and lowercase

print("Original Text:", text)
print("Cleaned Text:", clean_text)
print("Tokens:", tokens)
```

OUTPUT:



```
Original Text: I am learning NLP!!! It is fun, is not it?
Cleaned Text: I am learning NLP It is fun is not it
Tokens: ['i', 'am', 'learning', 'nlp', 'it', 'is', 'fun', 'is', 'not', 'it']
```

CONCLUSION: The program successfully performed text normalization by expanding contractions, cleaning, lowercasing, and tokenizing the text, and generated unigram, bigram, and trigram word combinations.

QUESTION

1. What is the purpose of removing special characters from a text during text normalization?

The purpose of removing special characters (e.g., punctuation marks, symbols like !!!, ?, @, #, $ etc.) from a text during text normalization is to:

- **Reduces Noise:** Special characters add noise without significant semantic meaning.
- **Improves Consistency:** Standardizes text by treating variations like "word." and "word" the same.
- **Focuses on Core Content:** Shifts focus to words and linguistic patterns for tasks like topic modeling.
- **Prepares for Tokenization:** Simplifies tokenization by removing complicating punctuation.

2. What is a bigram? Give an example from the sentence: "I love NLP."

A bigram (or 2-gram) is a sequence of two consecutive words from a text. It is a type of N-gram where N=2. Bigrams are used to capture local word order and context, which can be important for tasks like language modeling, text generation, and sentiment analysis.

Example from the sentence "I love NLP.":
The bigrams are:

("I", "love")
("love", "NLP")

EXP 5: POS Tagging

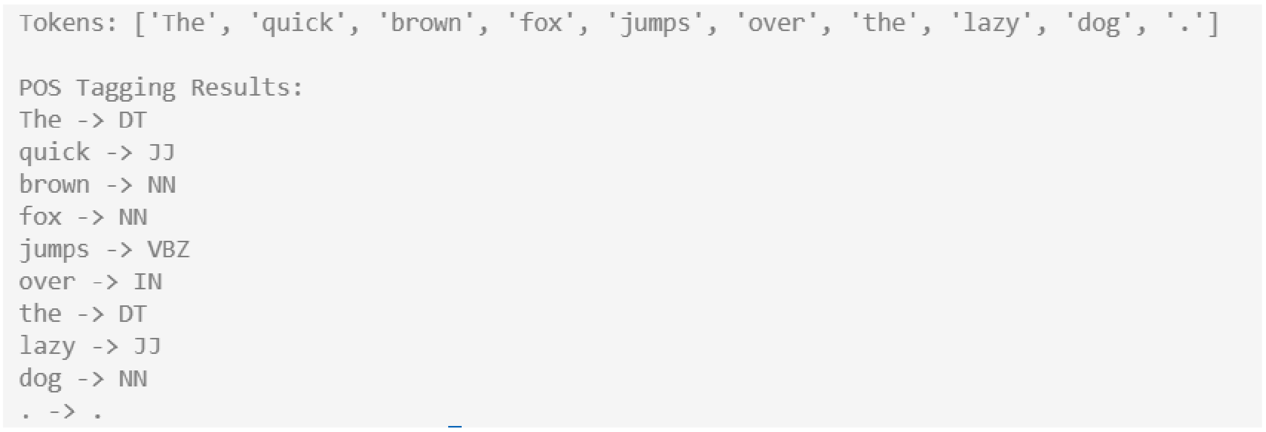
AIM: Write a program for POS tagging on the given text.

REQUIREMENTS(Software/Hardware):- Python Interpreter, Text Editor/IDE(VS Code), NLTK

PROGRAM:

```
import nltk
from nltk.tokenize import word_tokenize

nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')

text = "The quick brown fox jumps over the lazy dog."
tokens = word_tokenize(text)
pos_tags = nltk.pos_tag(tokens)

print("Tokens:", tokens)
print("\nPOS Tagging Results:")
for word, tag in pos_tags:
    print(f"{word} -> {tag}")
```

OUTPUT:



```
Tokens: ['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog', '.']

POS Tagging Results:
The -> DT
quick -> JJ
brown -> NN
fox -> NN
jumps -> VBZ
over -> IN
the -> DT
lazy -> JJ
dog -> NN
. -> .
```

CONCLUSION: The program successfully performed Part-of-Speech tagging using NLTK, assigning each word its grammatical label such as noun, verb, or adjective.

QUESTION

1. What is the role of the word_tokenize() function in the POS tagging program?

The wordtokenize() function's role in the POS tagging program is to break down the raw input text into individual words or tokens. Part-of-Speech (POS) tagging operates on individual words, not on the entire sentence as a single string. Therefore, word_tokenize() is a crucial preprocessing step that converts the continuous text into a list of discrete tokens, which can then be individually analyzed and assigned their respective grammatical tags.

2. Why do we use nltk.download('averaged_perceptron_tagger') before performing POS tagging?

We use nltk.download('averagedperceptrontagger') because it downloads the pre-trained model (specifically, an Averaged Perceptron Tagger model) that NLTK uses to perform Part-of-Speech tagging. POS tagging is a supervised learning task, and the tagger needs to have learned patterns from a large, annotated corpus to accurately assign tags to new words. This downloaded data contains the necessary linguistic rules and statistical information to predict the correct POS tag for each word. Without this model, NLTK would not be able to perform the tagging operation.