EXP:01 DATE: 05/01/2024

WORD ANALYSIS

AIM:

To implement word analysis using Python and NLTK.

ALGORITHM:

STEP1: Start the program

STEP2: Download Anaconda version 3.9

STEP3: Click Environment

STEP4: Create new environment with name Tensorflow and click Create

STEP5: Replace not installed option with installed

STEP6: Search tensorflow package and apply

STEP7: Repeat Step5, search Keras and NLTK and apply

STEP8: Go to home, click Tensorflow

STEP9: Install Spyder, Jupyter Notebook and launch

STEP10: String handling code has been generated and executed

STEP11: Stop the program

PROGRAM:

```
print(len("what it is what it isnt"))
s=["what","it","is","what","it","isnt"]
print(len(s))
x=sorted(s)
print(s)
print(x)
d=x+s
print(d)
```

OUTPUT:

```
print(len("what it is what it isn't"))
s=["what","it","is","what","it","isn't"]
print(len(s))
x=sorted(s)
print(s)
print(x)
d=x+s
print(d)
```

```
24
6
['what', 'it', 'is', 'what', 'it', "isn't"]
['is', "isn't", 'it', 'it', 'what', 'what']
['is', "isn't", 'it', 'it', 'what', 'what', 'what', 'it', 'is', 'what', 'it', "isn't"]
```

RESULT:

Word analysis using Python and NLTK is verified and executed.

EXP:02 DATE: 12/01/2024

WORD GENERATION

AIM:

To implement word generation using Python and NLTK.

ALGORITHM:

STEP1: Start the program

STEP2: Download Anaconda version 3.9

STEP3: Click Environment

STEP4: Create new environment with name Tensorflow and click Create

STEP5: Replace not installed option with installed

STEP6: Search tensorflow package and apply

STEP7: Repeat Step5, search Keras and NLTK and apply

STEP8: Go to home, click Tensorflow

STEP9: Install Spyder, Jupyter Notebook and launch

STEP10: File Handling is done using the process of tokenization and executed

STEP11: Stop the program

PROGRAM:

```
for line in open("nlp.py"):
    for word in line.split():
        if word.endswith('ing'):
          print(word)
          print(len(word))
```

OUTPUT:

```
[ ]  line="king"
```

```
for word in line:
    for word in line.split():
        if word.endswith("ing"):
            print(word)
            print(len(word))
```

```
king
4
king
4
king
4
king
4
```

RESULT:

Word generation using Python and NLTK is verified and executed.

EXP:03 DATE: 19/01/2024

MORPHOLOGY

AIM:

To implement morphology using Python and NLTK.

ALGORITHM:

STEP1: Start the program

STEP2: Download Anaconda version 3.9

STEP3: Click Environment

STEP4: Create new environment with name Tensorflow and click Create

STEP5: Replace not installed option with installed

STEP6: Search tensorflow package and apply

STEP7: Repeat Step5, search Keras and NLTK and apply

STEP8: Go to home, click Tensorflow

STEP9: Install Spyder, Jupyter Notebook and launch

STEP10: General Morphology Code and Stop Word Removal

STEP11: Stop the program

PROGRAM:

CODE:

```
import re

input="The5biggestanimalsare1.Elephant,2Rhinoand3dinasaur"

input=input.lower()

print(input)

result=re.sub(r'\d+','',input)

print(result)
```

STOP WORD REMOVAL:

```
def punctuations(raw_review):
  text=raw_review
  text=text.replace("n't",'not')
  text=text.replace("'s",'is')
  text=text.replace("'re",'are')
  text=text.replace("'ve",'have')
  text=text.replace("'m",'am')
  text=text.replace("'d",'would')
  text=text.replace("'ll",'will')
  text=text.replace("in",'ing')
  import re
  letters_only=re.sub("[^a-zA-Z]","",text)
  return(''.join(letters_only))
t="Hows'smyteamdoin ,you'resupposedtobenotloosin"
p=punctuations(t)
print(p)
```

SYNONYM:

```
import nltk
nltk.download('omw-1.4')
nltk.download('wordnet')
from nltk.corpus import wordnet
synonyms = []
for syn in wordnet.synsets('Machine'):
for lemma in syn.lemmas():
synonyms.append(lemma.name())
print(synonyms)
```

STEMMING:

```
from nltk.stem import PorterStemmer
stemmer=PorterStemmer()
print(stemmer.stem('eating'))
print(stemmer.stem('ate'))
```

OUTPUTS:

CODE:

```
import re
input="The 5 biggest animals are 1.Elephant,2 Rhinoand 3 dinasaur"
input=input.lower()
print(input)
result=re.sub(r'\d+','',input)
print(result)
```

```
the 5 biggest animals are 1.elephant,2 rhinoand 3 dinasaur
the  biggest animals are .elephant, rhinoand  dinasaur
```

STOP WORD REMOVAL:

```
[ ] def punctuations(raw_review):
        text = raw_review
        text = text.replace("n't", "not")
        text = text.replace("'s", "is")
        text = text.replace("'re", "are")
        text = text.replace("'ve", "have")
        text = text.replace("'m", "am")
        text = text.replace("'d", 'would')
        text = text.replace("'ll", 'will')
        text = text.replace("in", 'ing')
        import re
        letters_only = re.sub("[^a-zA-Z]", "", text)
        return(''.join(letters_only))

    t = "Hows'smyteamdoin ,you'resupposedtobenotloosin"
    p = punctuations(t)
    print(p)
```

```
Howsismyteamdoingyouaresupposedtobenotloosing
```

SYNONYM:

```
import nltk
nltk.download('omw-1.4')
nltk.download('wordnet')
from nltk.corpus import wordnet
synonyms = []
for syn in wordnet.synsets('Machine'):
    for lemma in syn.lemmas():
        synonyms.append(lemma.name())
print(synonyms)
```

```
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
['machine', 'machine', 'machine', 'machine', 'simple_machine', 'machine', 'political_machine', 'car', 'auto', 'automobile', 'machine', 'motorcar', 'machine', 'machine']
```

STEMMING:

```
[ ]  from nltk.stem import PorterStemmer
     stemmer=PorterStemmer()
     print(stemmer.stem('eating'))
     print(stemmer.stem('ate'))
```

```
eat
ate
```

RESULT:

The Morphological Analysis Code of NLP is verified and executed.

EX.NO:04 DATE: 02/02/2024

N-GRAMS

AIM:

To implement N-Grams using Python and NLTK.

ALGORITHM:

STEP1: Start the program

STEP2: Download Anaconda version 3.9

STEP3: Click Environment

STEP4: Create new environment with name Tensorflow and click Create

STEP5: Replace not installed option with installed

STEP6: Search tensorflow package and apply

STEP7: Repeat Step5, search Keras and NLTK and apply

STEP8: Go to home, click Tensorflow

STEP9: Install Spyder, Jupyter Notebook and launch

STEP10: N-Gram code has been generated and executed.

STEP11: Stop the program

PROGRAM:

```
import re

from nltk.util import ngrams

s ="Machine learning is an important part of AI""and AI is going to become inmporant for
daily functionong"

tokens=[token for token in s.split(" ")]

output =list(ngrams(tokens,2))

print(output)
```

OUTPUT:

```
import re
from nltk.util import ngrams
s = "Machine learning is an important part of AI and AI is going to become important for daily functioning"
tokens = [token for token in s.split(" ")]
output =list(ngrams(tokens,2))
display(output)
```

```
[('Machine', 'learning'),
 ('learning', 'is'),
 ('is', 'an'),
 ('an', 'important'),
 ('important', 'part'),
 ('part', 'of'),
 ('of', 'AI'),
 ('AI', 'and'),
 ('and', 'AI'),
 ('AI', 'is'),
 ('is', 'going'),
 ('going', 'to'),
 ('to', 'become'),
 ('become', 'important'),
 ('important', 'for'),
 ('for', 'daily'),
 ('daily', 'functioning')]
```

RESULT:

The N Grams code has been executed and verified using Python and NLTK.

EX.NO:05 DATE:09/02/2024

N-GRAMS SMOOTHING

AIM:

To implement N-Grams Smoothing using Python and NLTK.

ALGORITHM:

STEP1: Start the program

STEP2: Download Anaconda version 3.9

STEP3: Click Environment

STEP4: Create new environment with name Tensorflow and click Create

STEP5: Replace not installed option with installed

STEP6: Search tensorflow package and apply

STEP7: Repeat Step5, search Keras and NLTK and apply

STEP8: Go to home, click Tensorflow

STEP9: Install Spyder, Jupyter Notebook and launch

STEP10: N-Gram Smoothing code has been generated and executed.

STEP11: Stop the program

PROGRAM:

```
from collections import Counter

import numpy as np

# Define corpus

corpus = "the quick brown fox jumps over the lazy dog"

# Create unigrams

unigrams = Counter(corpus.split())
```

```
# Define function to compute n-grams
def get_ngrams(sentence, n):
    return [tuple(sentence[i:i+n]) for i in range(len(sentence)-n+1)]
# Create bigrams
bigrams = Counter(get_ngrams(corpus.split(), 2))
# Define smoothing function
def add_k_smoothing(ngram_counts, k, n_1gram_counts):
    # Calculate total number of n-grams
    total_ngrams = sum(ngram_counts.values())
    # Calculate vocabulary size
    vocabulary_size = len(n_1gram_counts)
    # Calculate denominator for probability calculation
    denominator = total_ngrams + k*vocabulary_size
    # Calculate smoothed probabilities
    probabilities = {}
    for ngram, count in ngram_counts.items():
        probabilities[ngram] = (count + k) / denominator
    # Handle unseen n-grams
    for ngram in set(n_1gram_counts.keys()) - set(ngram_counts.keys()):
        probabilities[ngram] = k / denominator
    return probabilities
# Apply smoothing to bigrams
k = 1
bigram_probabilities = add_k_smoothing(bigrams, k, unigrams)
# Print results
for bigram, probability in bigram_probabilities.items():
    print(bigram, probability)
```

OUTPUT:

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)

1161192
1161191
<FreqDist with 49815 samples and 1161192 outcomes>
69971
<FreqDist with 436003 samples and 1161191 outcomes>
258
251
81
44
2481534225
2,481,534,225
0.00017569896703721667
0.0002
```

RESULT:

The N-Gram Smoothing code has been executed and verified using Python and NLTK.

EX.NO:06 DATE:16/02/2024

POS – TAGGING: HIDDEN MARKOV MODEL

AIM:

To implement POS-Tagging: Hidden Markov Model using Python and NLTK

ALGORITHM:

STEP1: Start the program

STEP2: Download Anaconda version 3.9

STEP3: Click Environment

STEP4: Create new environment with name Tensorflow and click Create

STEP5: Replace not installed option with installed

STEP6: Search tensorflow package and apply

STEP7: Repeat Step5, search Keras, NLTK, Pandas, Numba and Random and apply

STEP8: Go to home, click Tensorflow

STEP9: Install Spyder, Jupyter Notebook and launch

STEP10:Using POS-Tagging, Hidden Markov Model code has been generated and executed.

STEP11: Stop the program

PROGRAM:

```
import nltk

import numpy as np
import pandas as pd
import random
from sklearn.model_selection import train_test_split
import pprint, time
 nltk.download('treebank')
nltk.download('universal_tagset')
 nltk_data = list(nltk.corpus.treebank.tagged_sents(tagset='universal'))
 print(nltk_data[:2]
```

OUTPUT:

```
[nltk_data] Downloading package treebank to /root/nltk_data...
[nltk_data]   Unzipping corpora/treebank.zip.
[nltk_data] Downloading package universal_tagset to /root/nltk_data...
[nltk_data]   Unzipping taggers/universal_tagset.zip.
[[('Pierre', 'NOUN'),
  ('Vinken', 'NOUN'),
  (',', '.'),
  ('61', 'NUM'),
  ('years', 'NOUN'),
  ('old', 'ADJ'),
  (',', '.'),
  ('will', 'VERB'),
  ('join', 'VERB'),
  ('the', 'DET'),
  ('board', 'NOUN'),
  ('as', 'ADP'),
  ('a', 'DET'),
  ('nonexecutive', 'ADJ'),
  ('director', 'NOUN'),
  ('Nov.', 'NOUN'),
  ('29', 'NUM'),
  ('.', '.')],
 [('Mr.', 'NOUN'),
  ('Vinken', 'NOUN'),
  ('is', 'VERB'),
  ('chairman', 'NOUN'),
  ('of', 'ADP'),
  ('Elsevier', 'NOUN'),
  ('N.V.', 'NOUN'),
  (',', '.'),
  ('the', 'DET'),
  ('Dutch', 'NOUN'),
  ('publishing', 'VERB'),
  ('group', 'NOUN'),
  ('.', '.')]]
```

RESULT:

Using POS Tagging, Hidden Markov Model has been executed and verified using Python and NLTK.

EX.NO:07 DATE:23/02/2024

POS – TAGGING: VITERBI DECODING

AIM:

To implement POS-Tagging: Viterbi Decoding using Python and NLTK

ALGORITHM:

STEP1: Start the program

STEP2: Download Anaconda version 3.9

STEP3: Click Environment

STEP4: Create new environment with name Tensorflow and click Create

STEP5: Replace not installed option with installed

STEP6: Search tensorflow package and apply

STEP7: Repeat Step5, search Keras and NLTK and apply

STEP8: Go to home, click Tensorflow

STEP9: Install Spyder, Jupyter Notebook and launch

STEP10:Using POS-Tagging, Viterbi Decoding has been generated and executed.

STEP11: Stop the program.

PROGRAM :

```
import nltk
from nltk.corpus import brown
# Training data
sentences = brown.tagged_sents()[:5000]
# Create tag frequency distribution and transition probability matrix
tag_freq = nltk.FreqDist(tag for sentence in sentences for word, tag in sentence)
transition_prob = nltk.ConditionalFreqDist(
    (tag1, tag2) for sentence in sentences for (_, tag1), (_, tag2) in nltk.bigrams(sentence)
)
```

```
# Define Viterbi function
def viterbi(sentence, tag_freq, transition_prob):
    # Initialize first word probabilities
    v = [{}]
    for tag in tag_freq:
        v[0][tag] = {"prob": tag_freq[tag] / len(sentences), "prev": None}
    # Recursion step
    for i in range(1, len(sentence)):
        v.append({})
        for tag in tag_freq:
            max_prob = max(
                v[i - 1][prev_tag]["prob"] * transition_prob[prev_tag][tag] * tag_freq[tag] /
len(sentences)
                for prev_tag in tag_freq
            )
            for prev_tag in tag_freq:
                if v[i - 1][prev_tag]["prob"] * transition_prob[prev_tag][tag] * tag_freq[tag] /
len(sentences) == max_prob:
                    v[i][tag] = {"prob": max_prob, "prev": prev_tag}
                    break
    # Termination step
    max_prob = max(value["prob"] for value in v[-1].values())
    current_tag = None
    for tag, data in v[-1].items():
        if data["prob"] == max_prob:
            current_tag = tag
            break
    # Backtracking
    tags = [current_tag]
    for i in range(len(v) - 1, 0, -1):
```

```
        current_tag = v[i][current_tag]["prev"]
        tags.append(current_tag)
    tags.reverse()
    return list(zip(sentence, tags))
# Example usage
sentence = "The quick brown fox jumps over the lazy dog".split()
pos_tags = viterbi(sentence, tag_freq, transition_prob)
print(pos_tags)
```

OUTPUT :

```python
import nltk
from nltk.corpus import treebank

# Train a POS tagger (you can replace treebank with your own tagged corpus)
train_data = treebank.tagged_sents()[:3000]
tagger = nltk.HiddenMarkovModelTagger.train(train_data)

# Define Viterbi decoding function
def viterbi_decode(sentence):
    return tagger.tag(sentence)

# Example usage
sentence = "This is a sample sentence."
tokenized_sentence = nltk.word_tokenize(se Loading...
tagged_sentence = viterbi_decode(tokenized_sentence)
print(tagged_sentence)
```

```
[('This', 'DT'), ('is', 'VBZ'), ('a', 'DT'), ('sample', 'JJ'), ('sentence', 'NNS'), ('.', '.')]
```

RESULT:

Using POS Tagging, Viterbi Decoding has been executed and verified using Python and NLTK.

EX.NO:08 DATE: 24/03/2024

BUILDING POS TAGGER

AIM:

To implement Building POS Tagger using Python and NLTK

ALGORITHM:

STEP1: Start the program

STEP2: Download Anaconda version 3.9

STEP3: Click Environment

STEP4: Create new environment with name Tensorflow and click Create

STEP5: Replace not installed option with installed

STEP6: Search tensorflow package and apply

STEP7: Repeat Step5, search Keras and NLTK and apply

STEP8: Go to home, click Tensorflow

STEP9: Install Spyder, Jupyter Notebook and launch

STEP10: Building POS Tagger has been generated and executed.

STEP11: Stop the program

PROGRAM:

```
import nltk
nltk.download('averaged_perceptron_tagger')
nltk.download('punkt')
text=nltk.word_tokenize("And now for Everything completely Same")
print(nltk.pos_tag(text))
```

OUTPUT :

```
[ ] import nltk
    nltk.download('averaged_perceptron_tagger')
    import nltk
    nltk.download('punkt')
    text = nltk.word_tokenize("And now for Everything completely Same")
    print(nltk.pos_tag(text))
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[('And', 'CC'), ('now', 'RB'), ('for', 'IN'), ('Everything', 'VBG'), ('completely', 'RB'), ('Same', 'JJ')]
```

RESULT:

Building POS Tagger code has been executed and verified using Python and NLTK.

EX.NO:09 DATE: 01/03/2024

CHUNKING

AIM:

To implement Chunking code using Python and NLTK

ALGORITHM:

STEP1: Start the program

STEP2: Download Anaconda version 3.9

STEP3: Click Environment

STEP4: Create new environment with name Tensorflow and click Create

STEP5: Replace not installed option with installed

STEP6: Search tensorflow package and apply

STEP7: Repeat Step5, search Keras and NLTK and apply

STEP8: Go to home, click Tensorflow

STEP9: Install Spyder, Jupyter Notebook and launch

STEP10: Chunking code is generated and verified, by printing the result.

STEP11: Stop the program

PROGRAM:

```
import nltk

sentence = [("the", "DT"), ("little", "JJ"), ("yellow", "JJ"), ("dog", "NN"), ("barked", "VBD"),
("at", "IN"), ("the", "DT"), ("cat", "NN")]

grammar = "NP: {<DT>?<JJ>*<NN>}"

cp = nltk.RegexpParser(grammar)

result = cp.parse(sentence)

print(result)

result.draw()
```

OUTPUT :

```python
import nltk

sentence = [("the", "DT"), ("little", "JJ"), ("yellow", "JJ"), ("dog", "NN"), ("barked", "VBD"),
("at", "IN"), ("the", "DT"), ("cat", "NN")]

grammar = "NP: {<DT>?<JJ>*<NN>}"
cp = nltk.RegexpParser(grammar)
result = cp.parse(sentence)
2333print(result)
```

```
(S
  (NP the/DT little/JJ yellow/JJ dog/NN)
  barked/VBD
  at/IN
  (NP the/DT cat/NN))
```

RESULT:

The chunking code has been executed and verified using Python and NLTK

EX.NO:10 DATE: 08/03/2024

BUILDING CHUNKERS

AIM:

To implement Building Chunkers code using Python and NLTK

ALGORITHM:

STEP1: Start the program

STEP2: Download Anaconda version 3.9

STEP3: Click Environment

STEP4: Create new environment with name Tensorflow and click Create

STEP5: Replace not installed option with installed

STEP6: Search tensorflow package and apply

STEP7: Repeat Step5, search Keras and NLTK and apply

STEP8: Go to home, click Tensorflow

STEP9: Install Spyder, Jupyter Notebook and launch

STEP10: Building Chunker code is generated and verified, by printing the result.

STEP11: Stop the program.

PROGRAM:

```
import nltk

from nltk.chunk import RegexpParser
# define chunking pattern
chunking_pattern = r"""
   NP: {<DT>?<JJ>*<NN>}  # noun phrase

      {<NNP>+}         # proper noun phrase
# tokenize and POS tag the text
text = "John saw the big brown bear in the forest"
tokens = nltk.word_tokenize(text)
pos_tags = nltk.pos_tag(tokens)

# apply chunking pattern to the POS tagged text
chunk_parser = RegexpParser(chunking_pattern)
chunks = chunk_parser.parse(pos_tags)
# print the extracted chunks rint(chunks)
```

OUTPUT:

```
import nltk
from nltk.chunk import RegexpParser

# define chunking pattern
chunking_pattern = r"""
    NP: {<DT>?<JJ>*<NN>}  # noun phrase
        {<NNP>+}           # proper noun phrase
"""

# tokenize and POS tag the text
text = "John saw the big brown bear in the forest"
tokens = nltk.word_tokenize(text)
pos_tags = nltk.pos_tag(tokens)

# apply chunking pattern to the POS tagged text
chunk_parser = RegexpParser(chunking_pattern)
chunks = chunk_parser.parse(pos_tags)

# print the extracted chunks
print(chunks)
```

```
(S
  (NP John/NNP)
  saw/VBD
  (NP the/DT big/JJ brown/NN)
  (NP bear/NN)
  in/IN
  (NP the/DT forest/NN))
```

RESULT:

Building Chunkers code has been executed and verified using Python and NLTK.