Expt. No. 01 Page No. 01

Expt. Name WORD ANALYSIS Date : 23.01.2024

Aim:- To perform basic word Analysis.

Procedure:-
For word analysis in python for NLP, we can use various libraries & Techniques. Here's a simple example using spacy to perform basic word analysis tasks. Spacy provides more sophisticated linguistic annotations and functionalities.

Tokenization:- The process of splitting the text into individual words or tokens.

Lemmentization:- Finding the base or dictionary form of a word

Dependency Parsing:- Analysing the syntactic structure of the sentence, showing relationships between words through dependency relations.

Algorithm:-
1. Import Spacy:- Import the Spacy library, which is an open-source natural language Processing Library

2. Load language Model:- Load the English language model provided by spacy, which includes a tokenizer, tagger, parser, named entity recognizer and word vectors.

3. Define Sample Text:- Set a sample text that will be used for analysis. On this case, its "NLP is a fascinating field of study".

4. Process text with Spacy:- Pass the sample text through the

Expt. No. _____ Page No. _____

Expt. Name _____ Date : _____

- Loaded spacy pipeline using nlp: This processes the text & generates a doc object that contains linguistic annotations & information about the text

5. Tokenization & Lemmatization :-
Extract tokens :- Get a list of all the tokens in the text using a list Comprehension
Lemmatization :- Get a list of lemmas using [token.lemma_ for toke in

6. Print tokens & lemmas :- Print the extracted tokens & their corresponding lemmas

7. Dependency Parsing :- Iterate through each tokens in the processed doc. Print information about each token's text, dependency relation, its head text, its head's parts of speech, & its children.

Expt. No. _____ Page No. _____

Expt. Name _____ Date : _____

Program :-

```
import spacy
nlp = spacy.load ("en_core_web_sm")
text = "Natural language Processing is a fascinating field of study."
doc = nlp(text)
tokens = [token.text for token in doc]
lemmas = [token.lemma_ for token in doc]
print ("Tokens:", tokens)
print ("Lemmas:", lemmas)
print ("\n Dependency Parsing:")
for token in Doc:
    print(token.text, token.dep_, token.head.text, token.head.pos_,
          [child for child in token.children])
```

Result :- Thus the expected output is successfully executed.

Output:-

Tokens: ['Natural', 'Language', 'Processing', 'is', 'a', 'fassinating', 'field', 'of', 'Stu

Lemmas: ['Natural', 'Language', 'Processing', 'be', 'a', 'fassinating', 'field', 'of', 'Stud

Dependency Parsing:

Natural Compound Language PROPN[]

Language Compound Processing PROPN[Natural]

Processing nsubj is AUX (Language)

is ROOT is AUX [Processing, field, .]

a det field NOUN[]

fascinating, amod field NOUN[]

field attr is AUX [a, fascinating, of]

of prep field NOUN [study]

Study pobj of ADP[]

. Punct is AUX[]

Expt. No. _____ Page No. 04

Expt. Name _____ Date : _____

Case Study:-

Aim:- To enhance Customer feedback Analysis through NLP-based text processing.

Problem statement:-
A company receives a large volume of customer feedback across various channels such as emails, social & surveys. Understanding & categorizing this feedback manually is time consuming & insufficient.

Objective :-
Utilize spacy & NLP techniques

Approach:

Data Collection:-

Text Preprocessing with spacy

Dependency Parsing Analysing

Insight Generallisation

Implementation:

Evaluation :

PROGRAM

Expt. No. _____ Page No. _____

Expt. Name _____ Date : _____

Analysing Feed back 2:-
The Customer feed back was terrible, very disappointed.
Tokens : [The Customer service was terrible , very disappointed .]
Lemmas: [The Customer Service was terrible ; very disappointed .]

Dependency Parsing:-
The det service NOUN[]
Customer Compound service NOUN[]
service nsubj was AUX [The, Customer]
Was ROOT was AUX [Service, disappointed, .]
terrible amod disappointed ADJ []
, Punct disappointed AD[]
.
very advmod disappointed ADJ[]
disappointed acmop was AUX [terrible, ,, very]
.punct was AUX []

Analysing feedback 3:-
Great Experience Overall
Tokens : [The GREAT Expericence overall]
Lemmas: [The GREAT Experience overall]

Dependency Parsing:-
GREAT amod experience NOUN[]
experience nsubj recommended VERB[Great]
overall advmod recommended VERB[]
, punct recommended VERB[]
highly advmod recommended VERB[].
! punct recommended VERB[]

Expt. No. _____

Expt. Name _____ Page No. _____

Date : _____

Conclusion:-

The developed NLP-based program utilizing Spacy moves to be an efficient solution for processing analysing customer feedback. Its capability to extract tokens, perform lemmetization & conduct dependency parsing aid in understanding the statement, identifing key topics & establishing relation ship with in feedback data

Result:-

This case study demonstrates the practical application of the provided code snippet using spacy in a bussiness context, specifically for customer feed back analysis. Showing how NLP techniques can be employed to extract valuble insights from unstructured text data.

Output:-

Tokens: [The product is ammazing ! I love the Quatity]

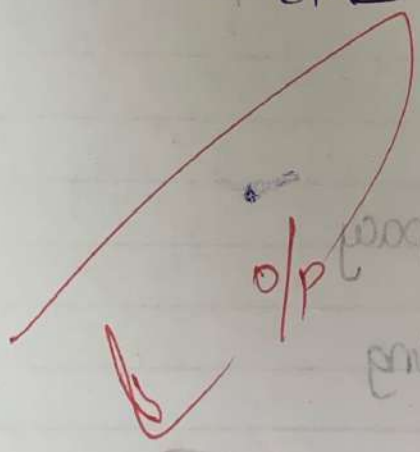
Tokens: [The Product is amazing ! I love the Quality]

Lemmas: [The product, be amazing ! I love the Quality]

Dependecy parsing:-

The det product NOUN]

Product nsubj is AUX. [The



Expt. No. 02 Page No. 07

Expt. Name WORD GENERATION Date : 30.01.2024.

Aim:- Word Generation using NLTK

Procedure:-

Loading Resources : Install NLTK if necessary & download Punkt & guten berg.

Loading Corpus : Load a specific Corpus from NLTK.

Bigram Generation: Create bigrams from the loaded Corpus to understand

Text Generation loop: Generate Text by selecting subsequent words based on the last word which is generated.

Random selection: Randomly choose the next word from the possibilities obtained.

Display Generated Text :- Print generated sequence of words

The following algorithm outlines the steps involved in generating text based on a Chosen Corpus using bigram model.

Expt. No. _____ Page No. _____

Expt. Name _____ Date : _____

ALGORITHM:-

1. Install & Import Libraries.
2. Download NLTK Resources
3. Load Corpus
4. Create Bigram Model
5. Choose starting Word
6. Generate Text
   - (a) Iterate 20 times
   - (b) For each Iteration
   - (c) Find all possible words
   - (d) Randomly select a word ~~from~~
   - (e) Append selected word.
7. Output Generated Text

PROGRAM:-

```
!PIP install NLTK
import nltk
import Random

nltk.download('Punkt')
nltk.download('gutenberg')
words = nltk.Corpus.gutenberg.words()
```

Expt. No. _____ Page No. _____

Expt. Name _____ Date : _____

```
bigrams = list(nltk.bigrams(words))
starting_word = 'the'.
generated_text = [starting_word]

for _ in range[20]:

possible_words = [word2 for (word1, word2) in bigrams if word1.
lower() = generated_text[-1].lower()]

next_word = random.choice(possible_words)
generated_text.append(next_word)

print(' '.join(generated_text))
```

Result:- The word generation is successfully executed and implemented.

Output :-

Requirement already satisfied

Requirement already satisfied

Requirement already satisfied

Requirement already satisfied

Requirement already satisfied

[nltk_data]. Downloading plunk

[nltk_data]. Package plunk

[nltk_data]. Downloading gutenberg

[nltk_data]. Package gutenberg.

the sea, avoided

o/p

Expt. No. 2 Page No. 10

Expt. Name CASE STUDY Date : 30.01.2024

Aim:- AutoComplete system for Email Composition

Problem Statement:-
A software company wants to develop an intelligent autoComp system for email composition. The goal is to assit users in generating coherent & contexually & contextually appropriate sentenses while composing emails.

Objectives:-

Approach:-

1. Data Collection:

2. Data Preprocessing:

3. Model Selection:

4. Context Integration:-

5. User Interface:

6. Model evaluation

7. Fine-Tuning & Iteration

Expt. No. _____ Page No. _____

Expt. Name _____ Date : _____

8. Deployment:-

Potential challenges:-

Context understanding

Ambiguity Handling

personalisation.

Success Criteria:-

Improved email composition efficiency & Speed.

Positive user feedback

Reducing typing errors.

By successfully developing & implementing this word generation program, the company aims to enhance to the productivity & user experience of individuals engaged in email communication.

Expt. No. _____ Page No. _____

Expt. Name _____ Date : _____

PROGRAM:-

Auto Complete Suggestions : ["How's", 'everything', 'going', 'on', '!', 'I', 'am', 'a', 'Programmer', 'and', 'I've', 'been', 'working', 'on', 'a', 'project', 'for', 'a', 'while', 'now', '.']

Enter your sentence (type 'exit' to end): exit.

Result :- The result demonstrates the integration of a powerful language model for enhancing user experience in composing emails through intelligent autocomplete suggestions.

Output:—

Enter your sentence (type 'exit' to end):
hello, howare you? How's every thing goingon!

The attention mask & the pad token id were not set. As a consequence, you may observe unexpected behaviour.

Setting 'Pad_token_id' to 'eos_token_id':
50256 for open-end generation.

o/p

Expt. No. 03 Page No. 13

Expt. Name Text classification Date : 06.02.2024

Aim:- To perform Text classification using python & scikit-it-learn

Procedure:-
This algorithm outlines the steps involved in the text classification task using linearSVC on the 20 Newsgroups dataset. It provides a structured approach

Algorithm:-

Algorithm : Text classification using Linear SVC

1. Load the 20 New groups dataset with specified categories.
① Import necessary libraries.
① Specify the categories of interest for classification
① use fetch_20 newsgroups

2. Split the dataset into training & testing sets.
① Import make train_test_split from sklearn.model_selection
① Split the dataset into X_train, X_test, y train & y test

3. Create a pipeline for text classification
① Import make_pipeline from sk.learn.
① Create a pipeline with TF-IDF vectorizer & linear SV

4. Train the model on the training data.
① Call the fit method on X_train & y_train as input.

5. Pr.

Expt. No. _____ Page No. _____

Expt. Name _____ Date : _____

5. Predict labels for the testing data.
⊙ Use the trained model to predict labels for X_test

6. Evaluate the model performance
⊙ Calculate accuracy score to measure
⊙ print classification

PROGRAM:-

```
!pip install scikit-learn
import pandas as pd

from sklearn.datasets
from sklearn.feature_extraction
from sklearn.pipeline
from sklearn.svm
from sklearn.model_selection
from sklearn.model_extraction

categories = ['sci.med', 'sci.space', 'comp.graphics', 'talk.politics.mideast

newsgroups_train = fetch_20newsgroups(subset='train', categories=cat

newsgroups_test = fetch_20newsgroups(subset='test', categories=categ

X_train = newsgroup_train data

X_test = newsgroup_test data
```

Expt. No. _____ Page No. _____

Expt. Name _____ Date : _____

```
y_train = newsgroup_train.target

y_test = newsgroups_test.target

model = make_pipeline(
TfidfVectorizer()
LinearSVC()
)

model.fit(X_train, y_train)

print("Accuracy:", accuracy)
print("\n Classification Report:")
print(Classification_report(y_test, predictions))
```

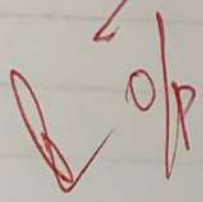
Result :- Thus the text is classified and executed the need full output successfully.

Output :-

Accuracy : 0.9504823151125402

Classification Report :-

	Precision	Recall	f1 score	support
0	0.89	0.97	0.93	309
1	0.96	0.91	0.94	396
2	0.98	0.94	0.96	394
3	0.98	0.98	0.96	376



Expt. No. 09 Page No. 16

Expt. Name Case Study Date : 06.02.2024

Aim:- Customer Support email classification

Problem Statement :-

A Customer Support Company recieves a large volume of incomming emails from customers with various inquiries, complaints & feedback.

Objectives :-

① The text classification system

① It improves the efficiency of Customer Support

① The Company can respond to customer

① Higher Customer satisfaction & retention.

Datasets :-

The Company has a dataset of past customer emails along with their corresponding categories. Each email is labe with one or more categories.

Approach :-

Data Preprocessing :-

①

①

Expt. No. _____ Page No. _____

Expt. Name _____ Date : _____

- Data Preprocessing
- Feature Extraction
- Model Selection
- Model evaluation
- Future Enhancements:

Program:-

```
from sklearn.datasets.impost fetch_20 newsgroups
from sklearn.feature_extraction
from sklearn.model-selection
from sklearn svm
from sklearn metrics impost accuracy score.
from
news groups = fetch_20 newsgroups (subset='all' ]

X = newsdatagroup

Y = newsgroup.target

X-train, X-test, Y-train, Y-test = train-test- split
                    (X, y, test_size = 0.2, random state = 42)
```

Expt. No. _____ Page No. _____

Expt. Name _____ Date : _____

```
Vectorizer = TfidfVectorizer(stop_words='english')
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)
classifier = LinearSVC()
classifier.fit(X_train, y_train)
predictions = classifier.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
print("Accuracy:", accuracy)
```

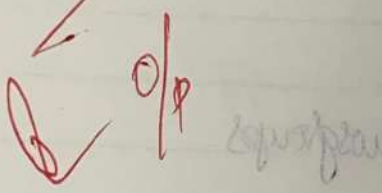
Result :- This case study outlines the problem statement, dataset, approach, expected outcome.

Output

Accuracy: 0.9389623601220752

Classification Report:-

	Precision	recall	f1-score	support
Comp. sys. ibm b.pc	0.92	0.91	0.91	212
Comp. sys. mac	0.94	0.93	0.94	198
rec. autos	0.97	0.93	0.95	179
rec. motorcycles	0.96	0.99	0.97	205
sci. electronics	0.92	0.93	0.92	189



Expt. No. 04 Page No. 19

Expt. Name Semantic Analysis Date 12.02.2024

Aim:- To perform Semantic Analysis using Gensim.

Procedure:-

Semantic analysis is a vast area in NLP. This program demonstrates semantic Analysis by leveraging pre-trained word vectors using word2vec from gensim.

Library installation

Library import
Pretrained word vectors
Sample Sentences
Tokenization
Semantic Analysis.

Algorithm:-

→ Install Necessary libraries

→ Import Libraries

→ Download Pretrained word vectors

→ Define Sample Sentences.

→ Tokenization

→ Semantic Analysis with word Vectors.

Expt. No. _____ Page No. _____

Expt. Name _____ Date : _____

Program:-

```
!Pip install gensim
!Pip install nltk

import gensim.downloader as api
from nltk.tokenize import word_tokenize

word_vectors = api.load ("word2vec-google-news-300")

sentences = ["Natural language Processing is a challenging but fasting field",
             "Word embeddings capture semantic meanings of words in a vs:"]

tokenized_sentences = [word_tokenize(sentence.lower()) for sentence in
                       sentences]
for tokenized_sentences in tokenized_sentences:
for word in tokenized_sentences:
    if word in word_vectors:
        similar_words = word_vectors.most_similar(word).
    print(f"Words similar to '{word}': {similar_words}")
    else:
        print(f"'{word}' is not in the pretrained word2vec Model.")
```

Result:-

Thus the semantic Analysis for the program is successfully executed & implemented.

Output:-

Requirement already satisfied :- gensim in /usr/local/lib/python 3.10/dist-packages (4.3.2)

Requirement already satisfied : numpy >= 1.18.5 in /usr/local/lib/python 3.10/dist-packages (from gensim) 1.23.

Expt. No. 04. Page No. 21

Expt. Name Case Study. Date : 13.02.2024.

Aim:-

Enhancing Customer service with semantic Analysis.

Problem Statement :-

A multinational e-commerce company "E-shop inc" is looking to improve its Customer service operations by leveraging advanced natural language processing techniques. They have a vast respitory of

Objectives:-

- Semantic Analysis
- Improving Customer service

Dataset:-

→ In the program, there isn't a specific dataset used for semantic analysis instead, the program semantic analysis on

Approach:-

- Tokenization
- Stop Removal
- Lemmatization
- Synonym Generation
- Output Generation

Expt. No. _____

Expt. Name _____ Page No. _____

Date : _____

Program:-

```
import nltk
from nltk.corpus import wordnet
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import wordnet lemmatizer)

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
def Semantic_analysis(text):
    tokens = word_tokenize(text):
    Stopwords = Set(stopwords.words('english'))
    filtered_tokens=[words for word in tokens if word.lower() not in
                     Stop_words].
    lemmatizer = WordNet lemmatizer()
    lemmalized_tokens=['lemmatizer.lemmatize(token) for token in
                       filtered_tokens]
    Synonyms = set()
    for token. in lemmatized_tokens:
        for syn in wordnet.synset(tokens)
        for lemma in sylemmas()
          Synonyms.add(lemma.name()).
        return list(Synonyms)
Customer_queries =[
"I recieved a damaged product can I get a refund?",
"I'm having trouble accessing my account.",
"how can I track my order status?"
"the item I recieved doesn't match the descripted,")
```

Expt. No. _____

Expt. Name _____ Page No. _____

Date : _____

```
for query in Customer Queries
    print ("Customer Query :" Query)
    Synonyms = Semantic anlysis (Query)
    print ("Semantic Analysis (Synonyms):", Synonyms)
    print("\n")
```

Result:-

Thus the program is sucessfully executed & output is generated Successfully.

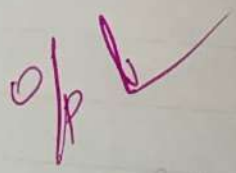
Output:-

[nltk_data] Downloading Package punkt to /root/nltk_data...

[nltk_data] unzipping tokenizers/punkt.zip.

[nltk_data] Downloading package stopwords to /root/nltk...

[nltk_data] unzipping corpora/stopwords.zip.



Expt. No. 05 Page No. 24

Expt. Name Sentiment Analysis. Date : 20.02.2024

Aim:-

To perform Sentiment analysis program using an SVM classifier with TF-IDF vectorization.

Procedure:-

Data Preparation

Splitting Data

TF-IDF Vectorization

SVM Initialization & Training

Prediction & Evaluation

Algorithm:-

Library Installation & Import

Download NLTK Resources

Load & Prepare dataset

Split Data into Train & Test set

TF-IDF vectorization

Initialize & Train SVM classifier

Prediction & Evaluation.

Expt. No. _____ Page No. _____

Expt. Name _____ Date : _____

Program:—

```
!pip install scikit-learn
!pip install nltk

import pandas as pd
from sklearn.model_selection import test_train_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
from sklearn.corpus import movie_review

import nltk
nltk.download('movie_review's)

documents = [(list(movie_reviews.words(fileid)), category)
        for category in movie_review.categories()
        for fileid in movie_reviews.field(category)]
    df = pd.DataFrame(document, columns=['text', 'sentiment'])

X_train, X_test, y_train, y_test = train_test_split(df['text'],
        df['sentiment'], test_size=0.2, random_state=42)

tfidf_vectorizer = TfidfVectorizer()

X_train_tfidf = tfidf_vectorizer.fit_transform(X_train apply
                                    ('.join))

svm_classifier = SVC(kernel='linear')
```

Expt. No. _____

Expt. Name _____ Page No. _____

Date : _____

```
svm_classifier.fit(X_train_tfidf, y_train)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

print(classification_report(y_test, y_pred))
```

Results-

Thus the Program is executed & implemented successfully

Output:-

Accuracy: 0.84

	Precision	recall	f-score	support
neg	0.83	0.85	0.84	199
Pos	0.85	0.82	0.84	201
accuracy			0.84	400
macro avg	0.84	0.84	0.84	400
weighted avg	0.84	0.84	0.84	400

o/p

Expt. No. 0
Expt. Name CASE STUDY
Page No. 21
Date: 20-02-2024

Aim:-

Sentiment Analysis of Product Reviews.

Problem Statement:-

A Company wants to analyze the sentiment analysis of customer Review for their Product. They want to understa whether reviews are positive, negitive & neutral.

Objectives:-

Understanding Customer Sentiment.

Identify Strengths & weakness.

Monitor customer Satisfaction

Inform Product Development.

Suppost Masketing Strategies.

Enhance Customer Service

Benchmask Performance

Validate Masket Reseach findings.

Dataset:-

Approach

Expt. No. _____ Page No. _____

Expt. Name _____ Date : _____

Program:-

```
import nltk
from nltk.sentiment.vader import sentimentIntensityAnalyzer.

nltk.download('vader_lexicon')

reviews =[
        "This Product is amazing! I love it.",
        "Netral Feedback on the product"]
    ]

sid = sentimentIntensityanalyzer()
for review in reviews:
    print("Review:", review)
    score = sid.polarity_scores(review)
print("Sentiment:", end=' ')
if scores['compound'] >0.05:
    print("Positive")
elsif scores['compound'] <-0.05:
    print("negitive")
else:
    print("neutral")
print()
```

Result:- This program demonstrates a basic Sentiment analysis using the VADER lexicon included in NLTK

Output:-

Review: This product is amazing! I love it.

Sentiment: Positive

Review: This product was good, but parking was damaged.

Sentiment: Negtive

Review: Natral feedback on the Product.

Sentiment: Neutral

O/P

Expt. No. 06

Expt. Name Parts Of Speech Tagging

Page No. 89

Date : 15-08-2024

Aim:-

To perform Parts of speech (POS) tagging using NLTK

Procedure:-

Libraries Installation & Import.

Download NLTK Resources

Sample text

Tokenization

POS Tagging

Display POS Tags.

Algorithm:-

Libraries installation & import

Download NLTK Resources.

Sample Text

Tokenization

POS Tagging & Display POS Tags

Expt. No. _____ Page No. _____

Expt. Name _____ Date : _____

Program :-

```
!Pip install nltk
import nltk
nltk.download('Punkt')
nltk.download('averaged_perceptron_tagger')

text = "Parts of speech tagging helps to understand the function
                                    of each word in a sentence."

tokens = nltk.word_tokenize(text)

Postags = nltk.pos_tags(tokens)

print("POS tags", Pos_tags)
```

Result :- Thus the program is installed & executed successfully with a required output

Output:-

Requirement already satisfied

[nltk_data] Downloading package punkt to /root/nltk_

POS tags: [('Parts', 'NNS'), ('of', 'IN'), ('Speech', 'NN'),
('the', 'DT'), ('function', 'NN'), ('of', 'IN'), (':', ':')]

Expt. No. 06. Page No. 81

Expt. Name CASE STUDY Date : 05-03-2024

Aim:-

Parts of speech Tagging

Algorithm:-

An online news aggregator wants to improve its recommendation system by analysing the content of news article

Objectives:-

Develop parts of speech tagging & analysing.

Extract the key information to understand structures.

Dataset:-

The data consists of collection of news article in text format - Each article is labled with its category.

Approach:-

Preprocess the dataset by tokenization.

Perform pos tagging using pre-trained model.

Extract relevent pos from tagged text

Analyze the & integrate extracted information

Expt. No. _____

Expt. Name _____ Page No. _____

Date : _____

Program:-

```
impost nltk.
from nltk.tokenize . impost word tokenize , sent tokenize.

nltk.download ('Punkt')
nltk.download ('averaged perceptron tagger')

def Pos taggeing (text):
    Sentences = sent_tokenize_(text)
    tagged tokens =[]
    for sentences in sentences
        tokens = word.tokenize (sentence)
        tagged_tokens.extend (nltk.pos_tag (tokens))
    return tagged tokens.

def main ():
    article text: """ Manchester unites secures a 3-1 victo
                    over chelsea in Yesterday Match. """
    tagged tokens = POS tagging (article text)
print
print
for tokens, POS tags in tagging_tokens:
    print (f"{tokens}: {POS tag}")

if __name__ == "__main__":
    main ()
```

Result:- Thus the program is executed & implemented Successfully

Outputs:-

Original Article Text:

"The Manchester Unites secures a 3-1 victory over chelsea in yesterday match."

Parts of speech tagging:

Manchester : NNP

Unites : NNP

secures : VBD

a : DT

3-1 : [illegible]

victory : NN

over : IN

Chelsea : NNP

in : IN

Yesterday : NN

Match : POS

'.' : .

Expt. No. 07 Page No. 33

Expt. Name CHUNKING Date : 12.03.2024

Aim:-

To perform Noun phrase chunking.

Procedure:-

In NLP, chunking is the process of Extracting short, m phrases from a sentence based on specific patterns of POS.

Tokenization
POS Tagging
chunking
chunk grammer
chunk praser

Algorithm:-

Import necessary libraries
Download nltk resources.
Define a sample sentence
Tokenization
POS
Chunk grammar definetion
Chunk phrase
Chunking
display chunks

Expt. No. _____ Page No. _____

Expt. Name _____ Date : _____

Program:-

```
!Pip install nltk
import nltk

from nltk import Regex Parser
from nltk tokenize import word_tokenize
from nltk.tag import word_tok POS_tag
from nltk.download ('Punkt')
    nltk.download ('averaged_perceptron_tagger')
sentence = "The Quick brown fox jumps over the lazy dogs"
tokens = words_tokenize (sentence)
tagged = POS tags (tokens)
chunk_grammer = r" "
  NP: {<DT>?<JJ>*<NN>}
  " " ".
Chunk_Praser = Regex Praser (Chunk_grammer)
Chunk = Chunk_Praser.Prase(tagged)
for subtree in chunk.subtrees():
  if subtree.label() == 'NP':
    print (Subtree).
```

Result:- Thus the Chunking is executed Successfully.

Output

Requirement already satisfied
Requirement already satisfies

[nltk-data]: Downloading Package Punkt to /root/nltk-data
[nltk-data]: Package Punkt is already up-to-date

0/1

Expt. No. _____

Expt. Name CASE STUDY

Page No. 35

Date : 12-03-2024

Aim:-

To demonstrate the extraction of the noun phrases from a given text using, chunking, a technique in NLP.

Problem Statement:-

Given a sample text, our goal is to identify & extract noun phrases, which are sequences of words containing a noun & optionally other words like adjectives or determiners.

Objectives:-

Tokenize input to text
perform POS tagging
Define & Apply chunking to extract & identify
Display the extracted noun phrases.

Dataset

For this Case study, we will use a sample text:
"The Quick brown fox jumped on the lazy dog".

Approach:-

The approach involves several steps to extract noun phrases from the given text using chunking in NLP.

Expt. No. _____

Expt. Name _____ Page No. _____

Date : _____

Program :-

```
import nltk
import os

nltk.data.Path.append("/usr/local/share/nltkdata")
nltk.download('Punkt')
nltk.download('averaged_perceptron_tagger')
text = "The Quick Brown fox jumps over the lazy dog"
words = nltk.word_tokenize(text)
pos_tags = nltk.POS_tags(words)
chunker grammer = r""
NP: {<DT>?<JJ>*<NN>}
chunk_parser = nltk.RegexParser(chunk_grammer)
chunked_text = chunk_parser.parse(pos_tags)
noun_phrases = []
for subtree in chunked_text.subtrees(filter = lam
                t.label() == 'NP'):
  noun_phrases.append(' '.join(word for word, tag
                subtree.leaves()))
print("Original text:", text)
print("Noun Phrases:")
  for phrases in noun_phrases:
    print("-", phrases).
```

Result :- Chunking is a valuble technique in NLP
identifing meaningful phrase from text.

Output

Original Text: The Quick brown fox jumps over the lazy dog.

Noun phrases:
The Quick brown fox
the lazy dog