INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

Dundigal - 500 043, Hyderabad, Telangana

COURSE CONTENT

NATURAL LANGUAGE PROCESSING LABORATORY								
VII Semester: CSE (AI&ML)								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
		L	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
ACAD30	**Core**	0	0	2	1	40	60	100
Contact Classes: NIL	**Tutorial Classes: NIL**	**Practical Classes: 45**				**Total Classes: 45**		
Prerequisite: **There are no prerequisites to take this course.**								

I. COURSE OVERVIEW:

This course is a study of computing systems that can process, understand, or communicate in human language. The primary focus of the course will be on understanding natural language processing (NLP) tasks and algorithms for effectively solving the problems using language models and evaluating the performance. NLP is most widely used as effective text processing techniques, strategies, and toolkits with a primary focus on sentiment analysis, pattern recognition, and chatbot development using libraries in the Python programming language.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The concepts and techniques of Natural language Processing for analyzing words based on Morphology and CORPUS.
II. The mathematical foundations and probability theory with linguistic essentials such as syntactic and semantic analysis of text.
III. The applications of statistical learning methods and cutting-edge research models from Artificial Intelligence.

III. COURSE OUTCOMES

After the completion of course the student is able to:

CO1	Summarize the concepts of complex behaviour in language to develop natural language processing applications.
CO2	Demonstrate the significance of words in a language using semantics and pragmatics to perform text processing.
CO3	Apply the CORPUS linguistics to compile and analyze the texts based on digestive approach (Text Corpus Method).
CO4	Compare statistical approaches like finding patterns and language acquisition for a given natural language to perform machine translation.
CO5	Analyze Part-of-speech tagging for a given natural language processing application to clear out the ambiguity in terms of revealing the grammatical structure.
CO6	Evaluate the performance of natural language-based systems for question-answering, text summarization, and machine translation with respect to morphology.

IV. COURSE CONTENT

EXERCISES FOR NATURAL LANGUAGE PROCESSING LABORATORY

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

1. Getting Started Exercises

Introduction, Installation and Accessing Additional Resources

Natural Language Toolkit (NLTK) is one of the vastest Python libraries for performing various Natural Language Processing tasks. From rudimentary tasks such as text pre-processing to tasks like vectorized representation of text – NLTK's API has covered everything. In this article, we will accustom ourselves to the basics of NLTK and perform some crucial NLP tasks: Tokenization, Stemming, Lemmatization, and POS Tagging.

Input: Python command to install NLTK library

Output: NLTK-An API library for performing an array of tasks related to human language.

Hint:

```
# Install the NLTK library
```

Try: Run the python script that incorporates the usage of additional resources of languages.

1.1 Tokenization

Perform the tokenization task which refers the breakdown of a text into smaller units. It entails splitting paragraphs into sentences and sentences into words. It is one of the initial steps of any NLP pipeline.

Input: Understand the major kinds of tokenization that NLTK provides.

Output: Word Tokenization.

Hint:

```
# Tokenization using NLTK
from nltk import word_tokenize, sent_tokenize sent =
"IARE is a great learning platform.\ It is one of the best
for AI&ML students."

# Print the tokens of a sentence here
```

Try: Use the same code and perform the sentence tokenization for any given sentence or a paragraph.

1.2 Stemming and Lemmatization

Understand the importance of meaning of the words by mapping every word of the language to its root/base form with the process called canonicalization.

Input: The words 'play', 'plays', 'played', and 'playing'.

Output: Mapping all the above words to their base form.

Hint:

```
from nltk.stem import PorterStemmer

# create an object of class PorterStemmer and print the stems of the words ‘play’, ‘playing’, ‘plays’, and ‘played’.
porter = PorterStemmer()
```

Try: Perform the grouping together using the concept lemmatization and produce the inflected forms of the same word. Implement the code to perform tokenization for the above same code.

1.3 Parts of Speech Tagging

Understand the importance of Parts of Speech of a word in a sentence and helps in giving a better syntactic overview of a sentence.

Input: “Institute of Aeronautical Engineering is the best college in Hyderabad”.

Output: Tags assigned by their POS tags.

Hint:

```
from nltk import pos_tag
from nltk import word_tokenize

text = "IARE is a Computer Science platform."
tokenized_text = word_tokenize(text)
tags = tokens_tag = pos_tag(tokenized_text)
```

Try: Implement the code to generate POS Tags to the words describing about the college IARE using one or two sentences.

1.4 POS Tagging and Lemmatization using spaCy

Understand the importance of text analysis and perform the information extraction task. Learn how to perform the extraction process and prepare the text for several deep learning applications.

Input: Install, import, and load spaCy library.

Output: POS Tags for words in “en_core_web_sm”

dataset **Hint:**

```
import spacy

# Load English tokenizer, tagger, #
parser, NER and word vectors nlp =
spacy.load("en_core_web_sm")

# Process whole documents
text = ("""My name is Shaurya Uppal.
I enjoy writing articles on GeeksforGeeks checkout my other
article by going to my profile section """)
```

```
doc = nlp(text) #

Token and Tag
for token in doc: print(token,
    token.pos_)

# You want list of Verb tokens
print("Verbs:", [token.text for token in doc if token.pos_ == "VERB"])
```

Try: Implement the code to perform lemmatization and process grouping together the inflected forms of a word so they can be analyzed as a single item, identified by the word's lemma, or dictionary form.

2. Text Processing in Python

2.1 Building NLP model and perform text processing.

Prepare the text data for the NLP model building and perform the text pre-processing. Use the required pre-processing steps based on the dataset prepared and understand the steps involved in Text Pre-processing.

Input: Few English statements

Output: Preprocessed Text - Implement the following text pre-processing operations.

a) Lowercase the text to reduce the size of the vocabulary of our text data

b) Remove numbers or convert the numbers into their textual representations

c) Remove punctuation so that we don't have different forms of the same word. If we don't remove the punctuation, then been. been, been! will be treated separately.

d) Use the join and split function to remove all the white spaces in a string.

Hint:

```
import pandas as pd import
numpy  as  np from sklearn
import svm
from sklearn import datasets
from sklearn.model_selection import train_test_split import
matplotlib.pyplot as plt
%matplotlib inline
        # Understand the dataset and write the code to evaluate the accuracy of a classification model
        …
```

Try: Develop a supervised machine learning model to identify the actual digits on a number plate.

2.2 Text Preprocessing Operations

Prepare the text data for the NLP model building and perform the text pre-processing. Use the required pre-processing steps based on the dataset prepared and understand the steps involved in Text Pre-processing. Implement the text pre-processing steps and perform various operations on word count.

Input: twitter-data/twitter4000.csv at master · laxmimerit/twitter-data · GitHub

Output: Preprocessed Text (Word Counts generated from several functions).

Hint:

```
import pandas as pd import numpy as np
import spacy from spacy.lang.en.stop_words import
STOP_WORDS as stopwords
df                                                                                    =
pd.read_csv('https://raw.githubusercontent.com/laxmimerit/twitterdata/master/twitte
r4000.csv', encoding = 'latin1') df

# Word Count
len('this is what nlp is'.split())
df['word_counts'] = df['twitts'].apply(lambda x: len(str(x).split())) df.sample(5)

# Max, Min, and One Word Count
df['word_counts'].max()
df['word_counts'].min()
df[df['word_counts']==1]

# Write the code for Character Count and Average word count
```

Try: Implement the code to perform other word count operations like:

a. Stopwords count.
b. Count Hash Tags and @Mentions.
c. Uppercase words count.

2.3 Preprocessing and Cleaning

Prepare the text data for the NLP model building and perform the text pre-processing. Use the required pre-processing steps based on the dataset prepared and understand the steps involved in Text Pre-processing. Implement the text pre-processing steps and perform various operations like:

a. Lowercase conversion
b. Contraction to expansion
c. Count and remove emails.

Input: twitter4000.csv file

Output: Text converted to lower case, expansions for contractions, and expressions without emails.

Hint:

```
# Lower Case Conversion
x = 'this is a sample Text' x.lower()
x = 45.0 str(x).lower()
df['twitts'] = df['twitts'].apply(lambda x: str(x).lower()) df.sample(5)

# Contraction to expansion
x = "i'm don't he'll" # "i am do not he will" def
cont_to_exp(x):
        if type(x) is str:
                for key in contractions:
                        value = contractions[key] x =
                        x.replace(key, value) return x
                else:
                        return x
        cont_to_exp(x) 'i am do not he will'
%%timeit
df['twitts'] = df['twitts'].apply(lambda x: cont_to_exp(x))
49.5 ms ± 375 µs per loop (mean ± std. dev. of 7 runs, 10 loops each) df.sample(5)

# Write the code to Count and remove emails
```

Try: Implement the code to perform the following text preprocessing operations:
a. Count URLs and remove them.
b. Remove RT
c. Remove special characters and punctuation.
d. Remove multiple spaces.

2.4 Preprocessing and Cleaning

Prepare the text data for the NLP model building and perform the text pre-processing. Use the required pre-processing steps based on the dataset prepared and understand the steps involved in Text Pre-processing. Implement the text pre-processing steps and perform various operations like:

a. Remove HTML Tags
b. Remove Accented Chars
c. Remove Stop words.

Input: twitter4000.csv file

Output: Text without HTML tags, accented chars, and stop words.

Hint:

```
# Remove HTML tags
!pip install beautifulsoup4. from bs4
import BeautifulSoup x = 'thanks for
watching it'
x.replace(",'') replace(",'') #not rec
```

```
BeautifulSoup(x, 'lxml').get_text().strip() %%time df['twitts'] =
df['twitts'].apply(lambda x: BeautifulSoup(x, 'lxml').get_text().strip())

# Remove accented chars x =
'Áccěntěd těxt' import
unicodedata
def remove_accented_chars(x):
        x = unicodedata.normalize('NFKD', x).encode('ascii', 'ignore').decode('utf-8', 'ignore')
        return x
remove_accented_chars(x)
df['twitts'] = df['twitts'].apply(lambda x: remove_accented_chars(x)) # Write the code

to remove stopwords
```

Try: Implement the code to perform the following text preprocessing operations:

a. Convert a word into its base form or root form.
b. Remove common words.
c. Remove rare words.

2.5 Preprocessing and Cleaning

Prepare the text data for the NLP model building and perform the text pre-processing. Use the required pre-processing steps based on the dataset prepared and understand the steps involved in Text Pre-processing. Implement the text pre-processing steps and perform various operations like:

a. Word Cloud visualization.
b. Spelling correction.
c. Tokenization using TextBlob

Input: twitter4000.csv file

Output: Visualization of word cloud, words with correct spellings, tokens.

Hint:

```
# Word Cloud Visualization #
!pip install wordcloud
from wordcloud import WordCloud import
matplotlib.pyplot as plt
%matplotlib inline
text = ' '.join(df['twitts']) len(text)
wc = WordCloud(width=800, height=400).generate(text)
plt.imshow(wc)
plt.axis('off')
plt.show()

# Write the code to perform Spelling Correction
```

```
# Tokenization using TextBlob
x = 'thanks#watching this video. please like it'
TextBlob(x).words doc
= nlp(x)
for token in doc:
        print(token)
```

Try: Implement the code to perform the following text preprocessing operations:

a. Detecting the nouns.
b. Language translation and detection.
c. Use TextBlob's Inbuilt sentiment analysis.

3. Regular Expressions

3.1 Built-in Modules for managing RE

Understand what regular expressions are and how we can leverage them for text feature engineering. Understand how REs are used in various data preprocessing tasks like information mining systems, text feature engineering, web scraping, and data extraction.

Input: Simple English sentences

Output: Preprocessed data suitable for text analysis.

Hint:

```
# ^ Matches the expression to its right, at the start of a string before it finds
a line finds a line break.
# $ Matches the expression to its left, at the end of a string before it finds a
line break
# . matches exactly one character a ab matches the string an

print(re.search(r"^.$",  "a"))  #  <re.Match  object;  span=(0,  1),
match='a'> print(re.search(r"^a$", "a")) # <re.Match object; span=(0,
1), match='a'> print(re.search(r"^.$", "hello")) # None
print(re.search(r"^he",  "hello"))  #  <re.Match  object;  span=(0,  2),
match='he'> print(re.search(r"lo$", "hello")) # <re.Match object; span=(3,
5), match='lo'>

# match function
print(re.search(r"hey.*", "hey, ciao
Fabio")) print(re.match(r"hey.*", "hey, ciao
Fabio")) print(re.search(r"ciao.*", "hey,
ciao Fabio")) print(re.match(r"ciao.*",
"hey, ciao Fabio"))

# findall and finditer functions
print(re.findall(r"\b\d+\b", "a 1 b 2 c 3"))
for match in re.finditer(r"\b\d+\b", "a 1 b 2 c
      3"): print(match)

# sub and subn functions
print(re.sub(r"\b\d+\b", "2022", "This year is 2021"))
print(re.subn(r"\b\d+\b", "NUMBER", "a 1 b 2 c 3 d 4", count=3))
```

```
# Write the code to use compile function and print the results using sub( ) and split(
) functions
```

Try: Implement the code how to use quantifiers:

- a|b : Matches expression a or b. If a is matched first, b is not checked;
- + : Matches the expression to its left 1 or more times;
- * : Matches the expression to its left 0 or more times;
- ? : Matches the expression to its left 0 or 1 times.
- {p} : Matches the expression to its left exactly p times;
- {p, q} : Matches the expression to its left p to q times;
- {p, } : Matches the expression to its left p or more times;
- {, q} : Matches the expression to its left up to q times.

3.2 Common Regex Functions used in NLP

Understand what regular expressions are and how we can leverage them for text feature engineering. Understand how REs are used in various data preprocessing tasks like information mining systems, text feature engineering, web scraping, and data extraction.

Input: Simple English sentences

Output: Preprocessed data suitable for text analysis.

Hint:

```
# Regular Expression import re
result = re.match("Institute of Aeronautical Engineering") print(result)
print(result.group())

# Replace XXX and YYY with abc in the given string string = "abc xxx abc
yyy"
new_string = re.sub(r"xxx|yyy", "abc",string)
print(new_string)

# Store the regular expression pattern in the cache memory pattern =
re.compile("Engineering")
result1 = pattern.findall("Institute of Aeronautical Engineering") print(result1)

# Write the code to return all the occurrences of the pattern from a string
```

Try: Implement the code to perform the following text preprocessing operations:

a. Sample regular expression to find quantifiers, anchors, and whitespace.

b. Sample regular expression to find wildcard characters, escaping special characters, and meta sequences.

3.3 Data Preprocessing using RE

Understand what regular expressions are and how we can leverage them for text feature engineering. Understand how REs are used in various data preprocessing tasks like information mining systems, text feature engineering, web scraping, and data extraction.

Input: Simple English sentences

Output: Preprocessed data suitable for text analysis.

Hint:

```
# Regular Expression import re
str="This is a sample text" pattern="This"
output=re.match(pattern,str).group(0) print(output)

import re
str="This is a sample text" pattern="sample"
output=re.search(pattern,str).group(0) print(output)
import re
str="This is a sample example with a sample text" pattern="sample"

# findall() searches for the RE and return a list after successful search output=re.findall(pattern,str)
print(output)

str="For example, 1-Jan-1900 is stored as number 1, 2-01-1900 is stored as 2, and 1-02-2015 is stored as 42005."
pattern=r'\d{1}-\d{2}-\d{4}'
# Write the code to represent the digits and the numbers in curly braces represents the number of digits
```

Try: Implement the code to perform the following text preprocessing operations:

a. Sample regular expression to find digits.
b. Wildcard expressions.

3.4 Anchors in Regular Expressions

Understand what regular expressions are and how we can leverage them for text feature engineering. Understand how REs are used in various data preprocessing tasks like information mining systems, text feature engineering, web scraping, and data extraction.

Input: Simple English sentences

Output: Preprocessed data suitable for text analysis.

Hint:

```
# Anchors in RE
import re
p = bool(re.search(r'\Acat', 'cater'))

# prefix \A to the search term
print(p)
p = bool(re.search(r'cat\Z', 'concatenation'))

# suffix \Z to restrict the match to the string end
print(p)
p = bool(re.search(r'\$hi', 'hi hello\ntop spot'))

# word boundary
print(p)
str = 'cats and dogs'
p = bool(re.search(r'^cat', str))

# Write the code to print the start and end of line

# ^ metacharacter for matching the start of line and $ for matching the end of
line print(p)
import re
str = 'institute of. aeronautical
engineering.' # without using backslash(\)
compare = re.search(r'.', str)
print(compare)

# using backslash(\)
compare = re.search(r'\.', str)

# Here, dot(.) is loosing its importance as a metacharacter
print(compare)
import re

# compile() creates regular expression # character class [a-
e], # which is equivalent to [abcde].
# class [abcde] will match with string
with # 'a', 'b', 'c', 'd', 'e'.
p = re.compile('[a-e]')
print(p.findall("interesting chunks to woodchunks and lemurs"))

# Disjunction, Grouping, and Precedence
str="This is the simple example with a sample text"
pattern="the"

# Write the code to use findall() to perform the search for the RE and return a
list after successful search


# Disjunction operator
pattern="dog|cat"
output=re.search(pattern,str) print(output)
import re
```

```
result=re.findall('cat|dog',r'cat and dog are domestic animals') print(result)
```

Try: Implement the code to perform the following text preprocessing operations:

a. Substitution of a specific text pattern.
b. Substituting a character set with a specific character.
c. Substitution up to a certain number of characters.

3.5 Grouping in Regular Expressions

Understand what regular expressions are and how we can leverage them for text feature engineering. Understand how REs are used in various data preprocessing tasks like information mining systems, text feature engineering, web scraping, and data extraction.

Input: Simple English sentences

Output: Preprocessed data suitable for text analysis.

Hint:

```
# Grouping in RE
import re
target_string = "The price of PINEAPPLE juice is 20"

# two groups enclosed in separate ( and ) bracket
result = re.search(r"(\b[A-Z]+\b).+(\b\d+)", target_string)

# Extract matching values of all groups
print(result.groups()) # Output ('PINEAPPLE', '20') # Extract match value of group
1 print(result.group(1)) # Output 'PINEAPPLE' # Extract match value of group 2
print(result.group(2)) # Output 20
Reference: https://pynative.com/python-regex-capturing-groups/
df

=
pd.read_csv('https://raw.githubusercontent.com/laxmimerit/twitterdata/master/twit
te r4000.csv', encoding = 'latin1')
import re
import pandas as
pd import numpy
as np import
spacy
df[df['twitts'].str.contains('hotmail.com')]
df.iloc[3713]['twitts'] x = '@securerecs arghh me please
markbradbury_16@hotmail.com' re.findall(r'([a-z0-9+._-]+@[a-z0-9+._-]+\.[a-z0-
9+_-]+)', x)
df['emails'] = df['twitts'].apply(lambda x: re.findall(r'([a-z0-9+._-] +@[a-z0-
9+._-
]+\.[a-z0-9+_-]+\b)', x))
```

```
# Here, lambda function takes any number of arguments but will have only one expression
# Write the code to retrieve student roll no's import re chat1='NLPbasics: Hello,
I am having an issue with my students with roll_nos 0513152435'
```

Try: Implement the code to retrieve the phone numbers and email ids from a given text/sentences.

4. Pattern Library in NLP

4.1 Parsing using Pattern

Understand the process of processing the text and different ways to analyze numeric data using Pattern library. Implement the code to perform natural language processing tasks like text mining by installing the pattern library using pip command.

Input: A simple English sentence.

Output: Tokens by setting the lemma parameter to TRUE.

Hint:

```
pip install pattern

from pattern.en import parse from
pattern.en import pprint
pprint(parse('Hello I am John, I work at the bank.', relations=True, lemmata=True))

# Add a parameter and increase the sentence length
pprint(parse('Enzo Ferrari was not initially interested in the idea of producing road cars when he formed
Scuderia Ferrari in 1929, with headquarters in Modena.', relations
= True,tokenize= True, lemmata= True))

# N-grams using Pattern
from pattern.en import ngrams #n
grams
print(ngrams("There is nothing either good or bad, but thinking makes it so.", n=3))

# Consider n=5 #n
grams
print(ngrams("There is nothing either good or bad, but thinking makes it so.", n=5))
```

Try: Implement the code to perform the sentiment analysis with Pattern library by understanding emotions and human sentiment from text data.

4.2 Word Corrections using Pattern

Understand how Pattern library inbuilt functions can assist in spelling correction and give a list of words as suggestions that may be the correct usage word. Implement the code to perform natural language processing tasks like text mining by installing the pattern library using pip command.

Input: A simple English token.

Output: Correct word with full confidence.

Hint:

```
#suggest
from pattern.en import suggest
print(suggest("Aerplane"))
# Try with a different word print(suggest(“Ambulnce”))
```

```
# Try a different case print(suggest("Cmputer"))

# Try with a different word
print(suggest("Entertanment"))

# Write the code to perform Word count estimation and quantification # Try with

different words
b = quantify(['Car', 'Car', 'Bus', 'Ambulance','Bus', 'Truck','Bus','Bus',
'Truck','FireTruck'])
print(b)

# Add numbers and see the result print(quantify({'Bus': 100, 'Car':
1500,'Truck': 700})) print(quantify('People', amount=60000))

# Add the numbers and see that the values of estimation increased
print(quantify({'Truck': 11, 'Car': 57808,'Bicycle': 564658}))
```

Try: Implement the code to try the function number and numerals that can convert numbers to words and vice versa.

4.2 Singular and Plural in Pattern

Understand how Pattern library inbuilt functions can assist in spelling correction and give a list of words as suggestions that may be the correct usage word. Implement the code to perform natural language processing tasks like text mining by installing the pattern library using pip command.

Input: A simple English token.

Output: POS of the words.

Hint:

```
from pattern.en import pluralize, singularize
print(pluralize('car')) print(singularize('BUSES'))
# Try with some different words
print(pluralize('Student')) print(singularize('chocolates'))

# Converting the adjective to comparative and superlative degrees from pattern.en
import comparative, superlative print(comparative('bad'))
print(superlative('bad'))

# Write the code to try a different word for the above task
```

Try: Implement the code to try Data Mining operations from Google search and Twitter that can be used to retrieve text data/information and make a certain process automated.

4.3 Tokenizing, POS Tagging, and Chunking

Understand how Pattern library provides an all-in-one method that takes a text string as an input parameter and returns the corresponding tokens in the string, along with the POS tag. Implement the code to retrieve the lemmatized tokens with the default values for different parameters.

Input: Use of Pattern library and parse function.

Output: Tokenized words along with their POS tags and lemmatized form of the tokens.

Hint:

```
# Implementation of parse method
parse(string,
      tokenize=True,          # Split punctuation marks from words? tags=True,
                              # Parse part-of-speech tags? (NN, JJ, ...) chunks=True,
                              # Parse chunks? (NP, VP, PNP, ...) relations=False,
                              # Parse chunk relations? (-SBJ, -OBJ, ...) lemmata=False,
                              # Parse lemmata? (ate => eat) encoding='utf-8', # Input
      string encoding.
      tagset=None             # Penn Treebank II (default) or UNIVERSAL.
)
from pattern.en import parse from
pattern.en import pprint

pprint(parse('I drove my car to the hospital yesterday', relations=True,
lemmata=True))

# Write the code to try with some different words and calling the split method

# Pluzalizing and singularizing the tokens from pattern.en
import pluralize, singularize

print(pluralize('leaf')) print(singularize('theives'))
```

Try: Implement the code to convert an adjective to comparative and superlative degrees.

4.4 Spelling Corrections using Pattern Library

Understand how Pattern library provides an all-in-one method that takes a text string as an input parameter and returns the corresponding tokens in the string, along with the POS tag. Implement the code to find if a word is spelled correctly or not and see how the 'suggest' function returns the possible corrections for the word along with their probability of correctness.

Input: Use of Pattern library and suggest function.

Output: Correct word spellings and results while working with numbers.

Hint:

```
# Spelling correction
from pattern.en import suggest print(suggest('Whitle'))

# Check for correct spelling print(suggest("Fracture"))

# Working with numbers
from pattern.en import number, numerals
print(number("one hundred and twenty two"))
print(numerals(256.390, round=2))

# Write the code to use the quantify function
```

Try: Implement the code to demonstrate the other word count estimators possible in natural language processing.

4.5 Pattern Library for Data Mining

Understand how Pattern library is used to perform a variety of data mining tasks. Implement the code to retrieve the contents from the web pages, extract URLs from text strings, and download the complete contents of the webpage etc.

Input: Webpage URL.

Output: Webpage

contents. **Hint:**

```
from pattern.web import download

page_html = download('https://en.wikipedia.org/wiki/Artificial_intelligence',
unicode=True)

from pattern.web import URL, extension

page_url =
URL('https://upload.wikimedia.org/wikipedia/commons/f/f1/RougeOr_football.jpg') file = open('football' +
extension(page_url.page), 'wb') file.write(page_url.download())
file.close()

# Finding URLs within Text
from pattern.web import find_urls
print(find_urls('To search anything, go to www.google.com', unique=True))

# Write the code to make Asynchronous Requests for webpages and print the results
```

Try: Implement the code to search something on Google via pattern library. Use the developer license key for the Google API.

5. Corpora or Corpus

5.1 Sample Usage of Corpus

Understand the importance of corpus reader classes that are used to access the contents of a diverse set of corpora. Implement the code to handle a specific corpus format and automatically create a set of corpus reader instances that can be used to access the corpora in the NLTK data package.

Input: Corpus reader with NLTK package.

Output: A list of identifiers obtained using a variety of methods provided by the corpus reader.

Hint:

```
import nltk.corpus

# The Brown corpus: print(str(nltk.corpus.brown).replace('\\\\','/'))

# The Penn Treebank Corpus: print(str(nltk.corpus.treebank).replace('\\\\','/'))

# The Name Genders Corpus: print(str(nltk.corpus.names).replace('\\\\','/'))

# The Inaugural Address Corpus:
print(str(nltk.corpus.inaugural).replace('\\\\','/'))

# Access list of identifiers nltk.corpus.treebank.fileids()
nltk.corpus.inaugural.fileids()

# Write the code to read data from the corpus and concatenate list of item names by loading all the documents in the corpus

# Access the README file inaugural.readme()[:32]
```

Try: Implement the code to extract the first few words from each of NLTK's plaintext corpora.

5.2 Tagged and Chunked Corpora

Understand the difference between plaintext corpora and tagged corpora. Implement the code to explore wide variety of annotated corpora using NLTK's data package.

Input: Brown Corpus with additional methods like tagged_*( ).

Output: List of words, tagged words, sentences, tagged sentences, categories of words etc.

Hint:

```
# Using the Brown Corpus
from nltk.corpus import brown print(brown.words())
print(brown.tagged_words()) print(brown.sents())
print(brown.tagged_sents())
print(brown.paras(categories='reviews'))
print(brown.tagged_paras(categories='reviews'))

# Using the Indian Corpus
from nltk.corpus import indian print(indian.words())
print(indian.tagged_words())

# Write the code to access the universal tag set and also the chunk structures
```

Try: Implement the code using IEER corpus to define the parsed_docs method and return the documents in each item as IEERDocument objects.

5.3 Text Normalization

Understand the importance of text normalization in the context of natural language processing. Implement the code to perform text processing like case normalization, punctuation removal, stop word removal, stemming, and lemmatization.

Input: Some sentences of English language.

Output: Normalized text

Hint:

```
# Text normalization code in Python
text = "The quick BROWN Fox Jumps OVER the lazy dog." text =
text.lower()
print(text)
# Punctuation Removal import
string
text = "The quick BROWN Fox Jumps OVER the lazy dog!!!"
text = text.translate(text.maketrans("", "", string.punctuation)) print(text)
# Stop word removal
from nltk.corpus import stopwords
text = "The quick BROWN Fox Jumps OVER the lazy dog." stop_words =
set(stopwords.words("english"))
words = text.split()
filtered_words = [word for word in words if word not in stop_words] text = "
".join(filtered_words)
print(text)

# Write the code to perform Stemming, Lemmatization, and Tokenization
```

Try: Implement the code to:

a. Replace synonyms and abbreviations to their full form to normalize the text in NLP.
b. Remove numbers and symbols to normalize the text.
c. Remove any remaining non-textual elements to normalize the text.

5.4 Minimum Edit Distance

Understand the concept of Minimum Edit Distance and implement the code to find the minimum number of edits (operations) required to convert one string into another string.

Input: Two strings of length M and N.

Output: Edit Distance between the strings **Hint:**

```
def edit_distance(str1, str2, a, b):
    string_matrix = [[0 for i in range(b+1)] for i in range(a+1)] for i in range(a+1):
        for j in range(b+1): if i ==
            0:
                string_matrix[i][j] = j
# If first string is empty, insert all characters of second string into first. elif j == 0:
                string_matrix[i][j] = i
# If second string is empty, remove all characters of first string. elif str1[i-1] ==
            str2[j-1]:
                string_matrix[i][j] = string_matrix[i-1][j-1]
# If last characters of two strings are same, nothing much to do. Ignore the last two characters and get
the count of remaining strings.
            else:
                string_matrix[i][j] = 1 + min(string_matrix[i][j-1], # insert
operation
                                              string_matrix[i-1][j],
# remove operation
                                              string_matrix[i-1][j-1])
# Continue the code to implement the replace operation for the above same task
```

Try: Implement the same code and test for different cases.

5.5 Minimum Edit Distance

Understand the concept of Minimum Edit Distance and implement the code to find the minimum number of edits (operations) required to convert one string into another string.

Input: Two strings of length M and N.

Output: Edit Distance between the strings

Hint:

```
import Levenshtein string1 =
"kitten"
```

```
string2 = "sitting"
edit_distance = Levenshtein.distance(string1, string2)
print("The edit distance between '{}' and '{}' is: {}".format(string1, string2, edi t_distance))

# Implement the code to use dynamic programming approach and finally print the edit distance
```

Try: Implement the same code and test for different cases.

6. Text to Speech Conversion

6.1 Perform the conversion of text to speech.

Using categorized CORPUS implement the code to perform the conversion of text to speech using gTTS library.

Input: API to complete the task

Output: Converted file into different languages.

Hint:

```
# Installing the gTTS API pip

install gTTS

# Install additional module pip
install playsound
pip install pyttsx3

# Start working with gTTS API
import gtts
from playsound import playsound

# Write the code to make a request to google to get synthesis, save the audio fule and play the same file
```

Try: Implement the same code and convert the text into other three different languages except English.

6.2 Text to Speech Conversion using Offline API

Using categorized CORPUS, implement the code to perform the conversion of text to speech using offline API and understand how to use pyttsx3 library.

Input: pyttsx3 library to complete the task

Output: Converted file into different languages.

Hint:

```
import pyttsx3

# Initialize Text-to-speech engine engine =
pyttsx3.init()

# Convert this text to speech
text = "Python is a great programming language" engine.say(text)

# Play the speech engine.runAndWait()

# Get details of speaking rate rate =
engine.getProperty("rate") print(rate)

# Write the code to pass the 100 to make the things slower and get details of all voices available
```

Try: Implement the code to perform the conversion of text to speech and build own virtual assistance.

6.3 Tokenization

Understand the concept of tokenization and learn how it affects the NLP pipeline. Implement the code to demonstrate how tokenization breaks unstructured data and natural language text into chunks of information that can be used directly as a vector representation of that document.

Input: Some sentences from English language

Output: Chunk of Tokens

Hint:

```
# Sentence Tokenization
Sent_tokenize('Life is a matter of choices, and every choice you make makes you.')

# Word Tokenization
word_tokenize("The sole meaning of life is to serve humanity")

# White Space Tokenization
Sentence = "I was born in India in 1980."
Sentence_split()

# Sentence Tokenization using comma as a separator
Sentence = "I was born in India in 1980, I am 43 years old".
Sentence_split(',')

# Write the code to perform work tokenize using NLTK package
```

Try: Implement the code to perform Treebank Word tokenization, Tweet tokenization, and MWET tokenization.

6.4 TextBlob Word Tokenization

Understand the concept of tokenization and learn how it affects the NLP pipeline. Implement the code to demonstrate how tokenization breaks unstructured data and natural language text into chunks of information that can be used directly as a vector representation of that document.

Input: Some sentences from English language

Output: Chunk of Tokens

Hint:

```
# Install installation TextBlob and the NLTK corpora pip install -
U textblob
python3 -m textblob.download_corpora

# Perform word tokenization using TextBlob library from textblob
import TextBlob
text = "But I'm glad you'll see me as I am. Above all, I wouldn't want people to think that I want to prove
anything".

blob_object = TextBlob(text)

#Word tokenization of the text
text_words = blob_object.words

#To see all tokens print(text_words)

# Write the code to count the number of tokens and perform the tokenization with keras
```

Try: Implement the code to perform spaCy Tokenization and Gensim word tokenization.

6.5 Text Normalization using Stemming.

Understand the concept of stemming and learn how stemming helps in normalizing the sentences that involves in reducing words to their root or base form. Implement the code to run the stemming process by removing suffixes or prefixes from words, so that related words map to the same stem. Also understand how Porter stemming algorithm works.

Input: Sample text to perform stemming

Output: Stems of the words.

Hint:

```
# Install NLTK library pip
install nltk
```

```
from nltk.stem import PorterStemmer from
nltk.tokenize import word_tokenize

# Initialize the Porter Stemmer porter_stemmer
= PorterStemmer()

# Sample text
text = "Stemming is a technique for reducing word forms to their base or root form."

# Write the code to perform tokenizing the text into words and perform stemming on each word

# Print the original and stemmed words
print("Original words:", words)
print("Stemmed words:", stemmed_words)
```

Try: Implement the same by adjusting the code according to the specific requirements and text data. Additionally, preprocess the text by converting it to lowercase or removing punctuation, depending on your application.

7. N-Gram Language Models

7.1 Unsmoothed Bigram Model

Suppose we didn't use the end-symbol. Train an unsmoothed bigram grammar on the following training corpus without using the end-symbol:

Input: <s> a b

<s> b b

<s> b a

<s> a a

Output: a) Demonstrate that your bigram model does not assign a single probability distribution across all sentence lengths by showing that the sum of the probability of the four possible 2 word sentences over the alphabet {a,b} is 1.0, and the sum of the probability of all possible 3 word sentences over the alphabet {a,b} is also 1.0.

Hint:

```
# Install the notebook as PDF Pip
install notebook-as-pdf

# Implementation of N-Grams # 1.
Exploring Data
# 2. Extracting features
# 3. Train and test set preparation # 4. Data
preprocessing
# 5. Generate n-grams
# 6. Generating unigrams, bigrams, trigrams

# Write the code to generate sentences from N-Grams
```

Try: Understand the above program and implement the code to achieve the solutions expected in the problem statement.

7.2 Language Models

Understand how n-grams are used to generate language models to predict which word comes next given a history of words. Implement the code to use 'lm' and 'nltk' libraries some sense of how natural language modelling is done.

Input: Corpus of your choice

Output: Prediction of best suitable words.

Hint:

```
# Write the code to generate sentences from corpus by using 7.1 exercise
```

Try: Implement the code to create a unigram language model using default dictionary.

7.3 Bigram Language Model

Understand how language models are used to generate or predict the exact word that comes next in each sentence. Develop the code to implement bigram language model.

Input: Corpus of your choice

Output: Predicting the best suitable next coming words in a sentence.

Hint:

```
# Create a function to train the bigram model def
train_bigram_model(data):

# Initialize a dictionary to store the counts counts =
defaultdict(lambda: defaultdict(int))

# Count the occurrences of each bigram for i in
range(len(data) - 1):
word1, word2 = data[i], data[i+1] counts[word1][word2]
+= 1

# Write the code to calculate the probabilities of each bigram # Example usage
text = "The quick brown fox jumps over the lazy dog" words =
preprocess(text)
bigram_model = train_bigram_model(words)
print(bigram_model)
```

Try: Implement the same code on different corpus and understand how the model preprocess function takes a list of words and returns them as preprocessed ones.

7.4 Trigram Language Model

Understand how language models are used to generate or predict the exact word that comes next in each sentence. Develop the code to implement trigram language model.

Input: Corpus of your choice

Output: Predicting the best suitable next coming words in a sentence.

Hint:

```
# Implementation of trigram language model import re
from collections import defaultdict
# Create a function to preprocess text data def
preprocess(text):
        # Convert all text to lowercase text =
        text.lower()
        # Remove all non-word characters text
        = re.sub(r'\W+', ' ', text)
        # Split the text into individual words words =
        text.split()
        return words

# Create a function to train the trigram model def
train_trigram_model(data):
        # Initialize a dictionary to store the counts
        counts = defaultdict(lambda: defaultdict(lambda: defaultdict(int))) # Count the
        occurrences of each trigram
        for i in range(len(data) - 2):
            word1, word2, word3 = data[i], data[i+1], data[i+2]
            counts[word1][word2][word3] += 1
        # Calculate the probabilities of each trigram probabilities =
        defaultdict(lambda: defaultdict(dict)) for word1 in counts:
            for word2 in counts[word1]:
                total_count = um(counts[word1][word2].values()) for
                word3 in counts[word1][word2]:
                    probabilities[word1][word2][word3] = counts[word1][word2][word3]
/ total_count return
probabilities

# Write the code to showcase the example usage
```

Try: Implement the same code on different corpus and understand how the model preprocess function takes a list of words and returns them as preprocessed ones.

7.5 Unigram, Bigram, and Trigram Language Models

Understand how n-grams are used to generate language models to predict which word comes next given a history of words. Implement the code to use 'lm' and 'nltk' libraries some sense of how natural language modelling is done.

Input: Corpus of your choice

Output: Prediction of best suitable words using 3 different language models.

Hint:

```
# import the libraries import
string
import random
import nltk
nltk.download('punkt') nltk.download('stopwords')
nltk.download('reuters')
from nltk.corpus import reuters from nltk
import FreqDist

# Input the reuters sentences sents
=reuters.sents()

# Write the removal characters such as : Stopwords and punctuation stop_words =
set(stopwords.words('english'))
string.punctuation = string.punctuation +''+''+'-'+''+''+'—' string.punctuation
removal_list = list(stop_words) + list(string.punctuation)+ ['lt','rt'] removal_list

# Write the code to generate unigrams bigrams trigrams


# Remove the n-grams with removable words def
remove_stopwords(x):
        y = []
        for pair in x: count
                = 0
                for word in pair:
                        if word in removal_list: count
                                = count or 0
                        else:
                                count = count or 1 if
                (count==1):
                        y.append(pair) return
        (y)
unigram = remove_stopwords(unigram)
bigram = remove_stopwords(bigram) trigram
= remove_stopwords(trigram)
```

```
# Generate frequency of n-grams freq_bi
= FreqDist(bigram) freq_tri =
FreqDist(trigram)

d = defaultdict(Counter) for a, b,
c in freq_tri:
        if(a != None and b!= None and c!= None): d[a, b]
           += freq_tri[a, b, c]


# Write the code to predict the next word for a given sentence
```

Try: Implement the same code by considering another corpus.

8. N-Gram Language Models

8.1 Analyzing different N-Gram Language Models

Analyze different types of n-grams on the given text data and decide which n-gram works better for your data.

Input: Text data

Output:
a) Implement the code by combining ngram taggers collectively using Unigram, Bigram and Trigram tagger.
b) Implement the code using backoff_tagger function for Unigram, Bigram and Trigram tagger and print proof as TRUE.

Hint:

```
# Import the Modules like cluster hierarchy, linkage, and dendrogram

import matplotlib.pyplot as plt import pandas
as pd

# Reading the DataFrame seeds_df

= pd.read_csv(
        "https://raw.githubusercontent.com/vihar/unsupervised-learning-with-
python/master/seeds-less-rows.csv")

# Remove the grain species from the DataFrame, save for later varieties =

list(seeds_df.pop('grain_variety'))

# Write the code to extract the measurements as a NumPy array and perform hierarchical
clustering on samples using the linkage () function with the method='complete' keyword
argument. Assign the result to mergings.
```

```
"""
Plot a dendrogram using the dendrogram () function on mergings, specifying the
keyword arguments labels=varieties, leaf_rotation=90, and leaf_font_size=6.

dendrogram(mergings,
                labels=varieties,
                leaf_rotation=90,
                leaf_font_size=6,
                )

plt.show()
```

Try: Implement the same code and do the performance comparison by considering newspaper and twitter corpus.

8.2 Smoothing Techniques in NLP

Understand the concept of smoothing techniques commonly used in NLP. Implement the code to build a N-gram language model and apply different smoothing methods to this language model and evaluate the results between these smoothing techniques. Do the comparison without and with smoothing techniques implementation.

Input: Default dictionary.

Output: Bigram language model without any smoothing technique implementation.

Hint:

```
from collections import defaultdict from
collections import Counter from
numpy.random import choice from tqdm
import tqdm

class Bigram():
    def _init_(self):
        self.bigram_counts = defaultdict(Counter)
        self.unigram_counts = Counter() self.context =
        defaultdict(Counter) self.start_count = 0
        self.token_count = 0
        self.vocab_count = 0

    def convert_sentence(self, sentence):
        return ["<s>"] + [w.lower() for w in sentence] + ["</s>"]

    def get_counts(self, sentences): #
        Collect unigram counts for
        sentence in sentences:
            sentence = self.convert_sentence(sentence) for word
            in sentence[1:]:
      # from 1, because we don't need the <s> token self.unigram_counts[word] +=
                1
            self.start_count += 1
```

```
        # Collect bigram counts for
        sentence in sentences:
            sentence = self.convert_sentence(sentence) bigram_list =
            zip(sentence[:-1], sentence[1:]) for bigram in bigram_list:
                self.bigram_counts[bigram[0]][bigram[1]] += 1
                self.context[bigram[1]][bigram[0]] += 1
        self.token_count = sum(self.unigram_counts.values())
        self.vocab_count = len(self.unigram_counts.keys())

    def generate_sentence(self):
        current_word = "<s>" sentence =
        [current_word] while current_word
        != "</s>":
            prev_word = current_word
            prev_word_counts = self.bigram_counts[prev_word]
            # Write the code to obtain bigram probability distribution given the previous word and
predict the sample the next word
```

Try: Implement the code to generate some sentences using the Penn Treebank corpora as training data.

8.3 Kneser-Ney Smoothing

Understand the concept of smoothing techniques commonly used in NLP. Implement the code to build a N-gram language model and apply different smoothing methods to this language model and evaluate the results between these smoothing techniques. Do the comparison without and with smoothing techniques implementation.

Input: Default dictionary.

Output: Bigram language model with kneser-ney smoothing technique implementation.

Hint:

```
def kneser_ney_smoothing(sentence, bigram, d): sentence =
    bigram.convert_sentence(sentence) bigram_list =
    zip(sentence[:-1], sentence[1:]) prob = 0

    for prev_word, word in bigram_list:
        sm_bigram_counts = bigram.bigram_counts[prev_word][word]
        if prev_word == "<s>": sm_unigram_counts = bigram.start_count else:
        sm_unigram_counts = bigram.unigram_counts[prev_word] if
        sm_unigram_counts == 0:
            prob += math.log((1 / float(bigram.vocab_count)) * 0.01) continue
        if sm_bigram_counts != 0:
            sm_bigram_counts = sm_bigram_counts - d else:
            # Write the code to perform the statistic how many tokens not occureed after pre_word
```

Try: Implement the code to generate some sentences using the Penn Treebank corpora as training data.

8.4 Morphological Analysis

Perform morphological analysis for an interrogative sentence, declarative sentence, and complex sentences with more than two sentences connected using conjunctions. Use the spaCy library and implement the code to perform the morphological word structure and forms.

Input: spaCy model

Output: Morphological analysis for interrogative sentence

Hint:

```
import spacy
nlp = spacy.load("en_core_web_sm")
interrogative_sentence = "What is the weather like today?"

# or interrogative_sentence = input("Enter an interrogative Sentence.") declarative_sentence = "The weather is sunny."

# or declarative_sentence = input("Enter an declarative Sentence.") complex_sentence = "I went to the store, but they were closed, so I had to go to another store."
# or complex_sentence = input("Enter an complex sentence using conjunction.")

interrogative_doc = nlp(interrogative_sentence)
declarative_doc = nlp(declarative_sentence) complex_doc
= nlp(complex_sentence)
 for token in interrogative_doc: print(token.text,
        token.pos_)
print("\n")
 for token in declarative_doc: print(token.text,
        token.pos_)
print("\n")
 for token in complex_doc: print(token.text,
        token.pos_)
```

Try: Develop the code to implement the morphological analyzer for Russian and English languages using dictionary-lookup systems.

8.5 Combining NGram Taggers

Analyze different types of n-grams on the given text data and decide which n-gram works better for your data.

Input: Text data

Output: Evaluation results

Hint:

```
# Working with bigram tagger #
Loading Libraries
```

```
from nltk.tag import DefaultTagger from
nltk.tag import BigramTagger

from nltk.corpus import treebank

# initializing training and testing set train_data =
treebank.tagged_sents()[:3000] test_data =
treebank.tagged_sents()[3000:]

# Write the code to perform Tagging for a bigram model and evaluate the same # Working of

Trigram Tagger
# Write the code to load libraries and initialize the training and testing set

# Tagging
tag1 = TrigramTagger(train_data)

# Evaluation
tag1.evaluate(test_data)

# Collectively using Unigram, Bigram and Trigram tagger. # Loading
Libraries

from nltk.tag import TrigramTagger from
tag_util import backoff_tagger from
nltk.corpus import treebank

# initializing training and testing set train_data =
treebank.tagged_sents()[:3000] test_data =
treebank.tagged_sents()[3000:]

backoff = DefaultTagger('NN') tag =
backoff_tagger(train_sents,
                                [UnigramTagger, BigramTagger, TrigramTagger], backoff
```

Try: Implement the same code using backoff_tagger function to create instances of each tagger class.

9. Word Sense Disambiguation

9.1 Lesk Algorithm

Understand the concept of word sense disambiguation and implement the code to involve the Lesk algorithm and count the number of words about the word and in the dictionary definition of that sense for each sense of the word being disambiguated.

Input: Dictionary of sentences

Output: Words count in the neighborhood of the word, pick the one with the greatest number of items in this count.

Hint:

```
%%capture
import nltk
from nltk.wsd import lesk
from nltk.tokenize import word_tokenize
nltk.download('all')

def get_semantic(seq, key_word):

        # Tokenization of the sequence temp =
    word_tokenize(seq)

    # Retrieving the definition # of
    the tokens
    temp = lesk(temp, key_word)
    return temp.definition()

# Sequence with the same word and different meanings keyword =
'book'
seq1 = 'I love reading books on coding.'
seq2 = 'The table was already booked by someone else.'

print(get_semantic(seq1, keyword))
print(get_semantic(seq2, keyword))

# Write the code to display output with two different sequences
```

Try: Implement the code to apply Lesk's algorithm using pywsd package for word-sense disambiguation by using WordNet implementation of Lesk.

9.2 Word Sense Disambiguation

Understand the concept of word sense disambiguation and implement the code to involve the Lesk algorithm and count the number of words about the word and in the dictionary definition of that sense for each sense of the word being disambiguated.

Input: Dictionary of sentences

Output: Words count in the neighborhood of the word, pick the one with the greatest number of items in this count.

Hint:

```
from nltk.wsd import lesk
from nltk.tokenize import word_tokenize

a1= lesk(word_tokenize('This device is used to jam the signal'),'jam')
print(a1,a1.definition())
a2 = lesk(word_tokenize('I am stuck in a traffic jam'),'jam') print(a2,a2.definition())

# testing with some data
```

```
b1= lesk(word_tokenize('Apply spices to the chicken to season it'),'season')
print(b1,b1.definition())
b2= lesk(word_tokenize('India receives a lot of rain in the rainy season'),'season')
print(b2,b2.definition())

# Write the code to perform testing with some data
```

Try: Develop the code to implement word sense disambiguation as a solution to the ambiguity that arises due to different meaning of words in different contexts.

9.3 Same Word, Different Meaning

In natural language processing, word sense disambiguation (WSD) is the problem of determining which "sense" (meaning) of a word is activated using the word in a particular context, a process which appears to be largely unconscious in people. WSD is a natural classification problem: Given a word and its possible senses, as defined by a dictionary, classify an occurrence of the word in context into one or more of its sense classes. The features of the context (such as neighboring words) provide evidence for classification.

Input: Corpus of your choice

Output: Print all the definitions of POS tags

Hint:

```
from nltk import wsd import
pandas as pd import numpy
as np import nltk
from nltk.corpus import wordnet as wn from
spacy.cli import download
from spacy import load
import warnings

nltk.download('omw-1.4')
nltk.download('wordnet')
nltk.download('wordnet2022') nlp =
load('en_core_web_sm')

! cp -rf /usr/share/nltk_data/corpora/wordnet2022
/usr/share/nltk_data/corpora/wordnet # temp fix for lookup error.

# in the below example the word die has a different meaning in each sentence.
# only by understanding the context the of the word the NLP can further improvise.

X = 'The die is cast.'
Y = 'Roll the die to get a 6.'
Z = 'What is dead may never die.

wn.synsets("die")
```

```
# check noun related details wn.synsets('die',
pos=wn.NOUN)

# Write the code to print all the definitions of nouns and verbs
```

Try: Implement automatic POS finding and correct definition using lesk. Before that will check a little about POS tagging using spicy.

9.4 Automatic POS Tagging + Lesk with spaCy

In natural language processing, word sense disambiguation (WSD) is the problem of determining which "sense" (meaning) of a word is activated using the word in a particular context, a process which appears to be largely unconscious in people. WSD is a natural classification problem: Given a word and its possible senses, as defined by a dictionary, classify an occurrence of the word in context into one or more of its sense classes. The features of the context (such as neighboring words) provide evidence for classification.

Input: Corpus of your choice
Output: Print all the definitions of POS tags

Hint:

```
POS_MAP = {
        'VERB': wn.VERB,
        'NOUN': wn.NOUN,
        'PROPN': wn.NOUN
}

def lesk(doc, word):
        found = False for
        token in doc:
                if token.text == word: word =
                        token
                        found = True
                        break
        if not found:
                raise ValueError(f'Word \"{word}\" does not appear in the document:
{doc.text}.')
        pos = POS_MAP.get(word.pos_, False) if
        not pos:
                warnings.warn(f'POS tag for {word.text} not found in wordnet. Falling back to default Lesk
behaviour.')
        args = [c.text for c in doc], word.text kwargs =
        dict(pos=pos)
       return wsd.lesk(*args, **kwargs) doc =
nlp('Roll the die to get a 6.') lesk(doc, 'die')

# Continue the code with finding POS tag by default, this helping lesk to find the correct definition.

# Check google as company.
lesk(nlp('I work at google.'), 'google').definition()
```

```
# check google as a verb (search engine)
lesk(nlp('I will google it.'), 'google').definition()

# hope as a noun
lesk(nlp('Her pep talk gave me hope'), 'hope').definition()

# hope as a veb
lesk(nlp('I hope we win!'), 'hope').definition()
```

Try: Develop the code to implement word sense disambiguation with Lesk algorithm.

9.5 POS – Text Processing

Implement the process of POS tagging by understanding the grammatical structure of a sentence. Develop the code by understanding the process of translation and information extraction where each word is related to each other in the sentence.

Input: en_core_web_sm dataset

Output: Apple | PROPN | proper noun | NNP noun, proper
singular is | AUX | auxiliary | VBZ verb, 3rd person singular
present
planning | VERB | verb | VBG verb, gerund or present
participle to | PART | particle | TO infinitival "to"
buy | VERB | verb | VB verb, base form
Indian | ADJ | adjective | JJ adjective (English), other noun-modifier
(Chinese) startup | NOUN | noun | NN noun, singular or mass
for | ADP | adposition | IN conjunction, subordinating or preposition
$ | SYM | symbol | $ symbol, currency
1 | NUM | numeral | CD cardinal number
billion | NUM | numeral | CD cardinal
number

Hint:

```
# Import Libraries pip
install spacy
python -m spacy download en_core_web_sm import

spacy

# Load the dataset
nlp = spacy.load("en_core_web_sm")

# Write the code to perform POS tagging on the text
```

Try: Implement the POS tagging in an NLP application of your choice and choose the best fits your needs.

10. Parts of Speech Tagging and Chunking

10.1 Word Classification

Understand how POS tagging process the words classification into their parts of speech and labelling them accordingly. Implement the code to understand how POS tagging is so important in understanding the meaning of a sentence or to extract the relationship and build a knowledge graph.

Input: Set of Words

Output:

a) Use part of speech tagging to mark a word to its part of speech tag based on its context in the data. It
is also used to extract relationships between words.

b) On top of Part of Speech tagging. It groups word into "chunks", mainly of noun phrases.
Chunking is done using regular expressions.

Hint:

```
# Write the code to create a classifier

# Import the text data to train the classifier # Define the

tags for your classifier

# Tag your data here

# Test and improve the text classifier
```

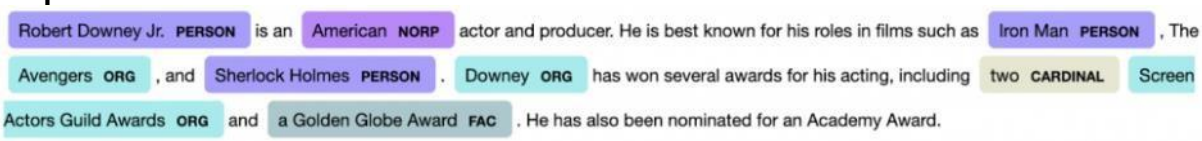
Try: Implement the code to help translate texts from one language to another by identifying the grammatical structure and relationships between words in the source language and mapping them to the target language.

10.2 Named Entity Recognition

Named Entity Recognition (NER) is an NLP technique to find and classify entities from textual data into predefined categories called named entities. This can include entities like an organization, an individual's name, location, a product, etc. Performing named entity recognition with a pre-trained model using Python.

Input: em_core_web_sm dataset

Output:



Hint:

```
# Choose the NER library

# Install the required libraries pip install
spacy
python -m spacy download en_core_web_sm

# Loading a pretrained model
import spacy
nlp = spacy.load("en_core_web_sm")

# Define the named entity categories that we want to recognize new_categories =
["PERSON", "ORG", "GPE", "PRODUCT"]

# Tokenizing the text
text = "Robert Downey Jr. is an American actor and producer. He is best known for his riles in films
such as iron man, The Avengers, and Sherlock Holmes. Downey has won several awards for his acting,
including two screen actors guild awards and a Golden Globe Award. He has also been nominated for
an Academy Award."
doc = nlp(text)

# Identifying and classification named entities entities = [ ]
for ent in doc.ents:
        if ent.label_ in ner_categories: entities.append((ent.text,
                ent.label_))
# Displaying the named entities and their categories for entity,
category in entities
        print(f"{entity}: {category}")
# Write the code to visualizing the results (entities in the document)
```

Try: Implement the code to explore the basics of NER and learn to use NER with pretrained models with python.

10.3 Named Entity Recognition using Spacy

Input: em_core_web_sm dataset

Output:

Hint:

```
# Import spacy
```

```
import spacy
from spacy import displacy

ner = spacy.load("en_core_web_sm") #

Collect the raw text
raw_text = "The Indian Space Research Organisation or is the national space agency of India, headquartered in Bengaluru. It operates under Department of Space which is directly overseen by the Prime Minister of India while Chairman of ISRO acts as executive of DOS as well."

text1= NER(raw_text)

# Write the code to print the data on the named entity
```

Try: Implement the code to try the same tasks with some tests containing more named entities.

raw_text2="The Mars Orbiter Mission (MOM), informally known as Mangalyaan, was launched into Earth orbit on 5 November 2013 by the Indian Space Research Organisation (ISRO) and has entered Mars orbit on 24 September 2014. India thus became the first country to enter Mars orbit on its first attempt. It was completed at a record low cost of $74 million."

10.4 Byte Pair Encoding

Byte Pair Encoding (BPE) is a widely used tokenization method among transformer-based models. BPE is a word segmentation algorithm that merges the most frequently occurring character or character sequences iteratively. Implement the code to apply BPE to OOV words.

Input: Sample text of your choice

Output: Frequency of each word in the corpus

```
Corpus: {'l o w </w>': 5, 'l o w e r </w>': 2, 'n e w e s t </w>': 6, 'w i d e s t </w>': 3}
```

Hint:

```
# Reading the corpus
# importing the libraries import
pandas as pd

# reading .txt file
text = pd.read_csv("sample.txt", header = None)

# converting a dataframe into a single list corpus = [ ]
for row in text.values:
        tokens = row[0].split(" ") for token
        in tokens:
                corpus.append(token)

vocab = list(set(" ".join(corpus))) vocab.remove(' ')

# split the word into characters
```

```
corpus = [“ “.join(token) for token in corpus]

# appending </w>
corpus = [token+’</w>’ for token in corpus]
print(corpus)

import collections

# Write the code to return the frequency of each word

# Write the code to convert counter object to dictionary and print the same
```

Try: Implement the code to compute the frequency of a pair of character or character sequences by accepting the corpus and return the pair with its frequency.

10.5 BERT Named Entity Recognition (NEW)

Input: BERT Uncased dataset

Output: Predicted named entities and their labels

Hint:

```
import tensorflow as tf import
tensorflow_hub as hub

# Load a BERT model from TensorFlow Hub
model = hub.load("https://tfhub.dev/google/bert_uncased_L-12_H-768_A-12/1")

# Define a sample text
text = "Barack Obama was the 44th president of the United States."

# Tokenise and input the text to the BERT model tokens =
model.tokenization.tokenize_string(text)
inputs = model.tokenization.convert_tokens_to_ids(tokens)

# Use the BERT model to predict named entities ner_predictions =
model.predict(inputs)

# Write the code to iterate over the predicted named entities and print their labels
```

Try: Use the concept of NER and implement the code to perform the entity recognition using FLAIR library.

11. Text Analysis

11.1 Real-Time Twitter Sentiment Analysis

Carefully listening to the voice of the customer on Twitter using sentiment analysis allows companies to understand their audience, keep on top of what’s being said about their brand – and their competitors –

and discover new trends in the industry. The goal of this exercise is to understand how you can use sentiment analysis tools to listen to your customers on Twitter.

Input: Twitter dataset

Output: Customer

Sentiments **Hint:**

```
# import the necessary dependencies import
re
import numpy as np
import pandas as pd #
plotting
import seaborn as sns
from wordcloud import WordCloud
import matplotlib.pyplot as plt # nltk
from nltk.stem import WordNetLemmatizer #
sklearn
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import BernoulliNB
from sklearn.linear_model import LogisticRegression from
sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import confusion_matrix, classification_report

# Read and load the datasets #
Importing the dataset
DATASET_COLUMNS=['target','ids','date','flag','user','text']
DATASET_ENCODING = "ISO-8859-1"
df = pd.read_csv('Project_Data.csv', encoding=DATASET_ENCODING,
names=DATASET_COLUMNS)
df.sample(5)

# Exploratory Data Analysis
df.head()

df.columns
print('length of data is', len(df))

df.    Shape
df.info()
df.dtypes
np.sum(df.isnull().any(axis=1))

print('Count of columns in the data is:  ', len(df.columns)) print('Count of
rows in the data is:  ', len(df))

df['target'].unique()
df['target'].nunique()

# Data Visualization of Target Variables # Plotting
the distribution for dataset.
```

```
ax.set_xticklabels(['Negative','Positive'],
rotation=0) # Storing data in lists.
text, sentiment = list(df['text']), list(df['target'])

import seaborn as sns
sns.countplot(x='target', data=df)

# Data Preprocessing
data=df[['text','target']]
data['target'] = data['target'].replace(4,1)
data['target'].unique()

data_pos = data[data['target'] ==
1] data_neg = data[data['target']
== 0]

data_pos = data_pos.iloc[:int(20000)]
data_neg = data_neg.iloc[:int(20000)]

dataset = pd.concat([data_pos, data_neg])

dataset['text']=dataset['text'].str.lower()
dataset['text'].tail()

# Write the code to include the list of stopwords


STOPWORDS =
set(stopwordlist) def
cleaning_stopwords(text):
    return " ".join([word for word in str(text).split() if word not in STOPWORDS])
dataset['text'] = dataset['text'].apply(lambda text: cleaning_stopwords(text))
dataset['text'].head()

import string
english_punctuations =
string.punctuation punctuations_list =
english_punctuations def
cleaning_punctuations(text):
    translator = str.maketrans('', '', punctuations_list)
    return text.translate(translator)
dataset['text']= dataset['text'].apply(lambda x: cleaning_punctuations(x))
dataset['text'].tail()

def cleaning_repeating_char(text):
    return re.sub(r'(.)1+', r'1',
    text)
dataset['text'] = dataset['text'].apply(lambda x: cleaning_repeating_char(x))
dataset['text'].tail()

def cleaning_URLs(data):
    return re.sub('((www.[^s]+)|(https?://[^s]+))',' ',data)
dataset['text'] = dataset['text'].apply(lambda x: cleaning_URLs(x))
dataset['text'].tail()

def cleaning_numbers(data):
    return re.sub('[0-9]+', '', data)
dataset['text'] = dataset['text'].apply(lambda x: cleaning_numbers(x))
dataset['text'].tail()
```

```
from nltk.tokenize import RegexpTokenizer tokenizer =
RegexpTokenizer(r'w+')
dataset['text'] = dataset['text'].apply(tokenizer.tokenize) dataset['text'].head()

# Applying Stemming
import nltk
st = nltk.PorterStemmer() def
stemming_on_text(data):
        text = [st.stem(word) for word in data] return data
dataset['text']= dataset['text'].apply(lambda x: stemming_on_text(x)) dataset['text'].head()

# Write the code to apply Lemmatization

# Splitting the data into training and testing
# Separating the 95% data for training data and 5% for testing data X_train, X_test, y_train,
y_test = train_test_split(X,y,test_size = 0.05, random_state =26105111)

# Data Transformation using TF-IDF Vectorizer

vectoriser = TfidfVectorizer(ngram_range=(1,2), max_features=500000)
vectoriser.fit(X_train)
print('No. of feature_words: ', len(vectoriser.get_feature_names()))

X_train = vectoriser.transform(X_train) X_test =
vectoriser.transform(X_test)

# Write the code to evaluate the model and predict the values for the test dataset # Write the code to

compute and plot the Confusion matrix
```

Try: Implement the code to extend the above problem statement by building the model and plotting the results.

12. Markov Model Process

12.1 Word Similarity

Examine how Word Similarity is used to determine how semantically two words are close to each other. Find the similarity between the word vectors in the vector space using spaCy NLP library.

Input: Sample set of words

Output:

a) Find word similarity: using context-sensitive tensors.

b) Determine the type of an image in Python using imghdr.

Hint:

```
{
```

```
 "cells": [
  {
   "cell_type":
   "markdown",
   "metadata": {},
   "source": [
    "## What is Reinforcement Learning?"
   ]
  },
  {
   "cell_type":
   "markdown",
   "metadata": {},
   "source": [
    "Consider you are teaching the dog to catch a ball, but you cannot teach the
dog explicitly to\n",
    "catch a ball, instead, you will just throw a ball, every time the dog catches
a ball, you will\n",
    "give a cookie. If it fails to catch a dog, you will not give a cookie. So
the dog will figure out\n",
    "what actions it does that made it receive a cookie and repeat that action."
   ]
  },
  {
   "cell_type":
   "markdown",
   "metadata": {},
   "source": [
    "Similarly in an RL environment, you will not teach the agent what to do or
how to do,\n",
    "instead, you will give feedback to the agent for each action it does.
The feedback may be\n",
    "positive (reward) or negative (punishment). The learning system which
receives the\n",
    "punishment will improve itself. Thus it is a trial and error process.
The reinforcement\n",
    "learning algorithm retains outputs that maximize the received reward over
    time.
In the\n",
    "above analogy, the dog represents the agent, giving a cookie to the dog
on catching a ball is\n",
    "a reward and not giving a cookie is punishment.\n",
    "\n",
    "There might be delayed rewards. You may not get a reward at each step. A
reward may be\n",
    "given only after the completion of the whole task. In some cases, you get
a reward at each\n",
    "step to find out that whether you are making any mistake.\n",
    "\n",
    "An RL agent can explore for different actions which might give a good reward
or it can\n",
    "(exploit) use the previous action which resulted in a good reward. If the RL
agent explores\n",
    "different actions, there is a great possibility to get a poor reward. If the
RL agent exploits\n",
    "past action, there is also a great possibility of missing out the best
action which might give a\n",
    "good reward. There is always a trade-off between exploration and
    exploitation.
We cannot\n",
```

```
    "perform both exploration and exploitation at the same time. We will
discuss exploration exploitation\n",
    "dilemma detail in the upcoming
    chapters.\n", "\n",
    "Say, If you want to teach a robot to walk, without getting stuck by hitting
at the mountain,\n",
    "you will not explicitly teach the robot not to go in the direction of
    mountain,"
]
},
  {
   "cell_type":
   "markdown",
   "metadata": {},
   "source": [
    "![title](images/B09792_01_01.png)"
]
},
  {
   "cell_type":
   "markdown",
   "metadata": {},
   "source": [
    "Instead, if the robot hits and get stuck on the mountain you will reduce
10 points so that\n",
    "robot will understand that hitting mountain will give it a negative reward
so it will not go\n",
    "in that direction again."
   ]
  },
  {
   "cell_type":
   "markdown",
   "metadata": {},
   "source": [
    "![title](images/B09792_01_02.png)"
   ]
  },
  {
   "cell_type":
   "markdown",
   "metadata": {},
   "source": [
    "And you will give 20 points to the robot when it walks in the right
direction without getting\n",
    "stuck. So robot will understand which is the right path to rewards and try to
maximize the\n",
    "rewards by going in a right direction."
]
},
  {
   "cell_type":
   "markdown",
   "metadata": {
   "collapsed": true
   },
   "source": [
    "![title](images/B09792_01_03.png
    )"
   ]
```

```
  }
 ],
```

```
 "metadata": {
  "kernelspec": {
   "display_name": "Python [conda env:anaconda]", "language":
   "python",
   "name": "conda-env-anaconda-py"
  },
  "language_info": {
   "codemirror_mode": { "name":
   "ipython", "version": 2
   },
   "file_extension": ".py", "mimetype":
   "text/x-python", "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython2", "version":
   "2.7.11"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 2
}
```

Try: Implement the code to clean the text data and apply Markov chains for text generation.

13. POS Tagging

13.1 Generating Knowledge Graph

Learn how POS tagging works by describing words in their parts of speech and help you understand the meaning of sentences and develop a knowledge graph.

Input: Some set of words

Output: POS of each

word **Hint:**

```
import spacy
nlp = spacy.load('en_core_web_sm')

doc = nlp("The 22-year-old recently won ATP Challenger tournament.") for tok in

doc:
    print(tok.text, "...", tok.dep_)

doc = nlp("Nagal won the first set.") for tok in
doc:
        print(tok.text, "...", tok.dep_) # import

libraries

# Read the CSV file containing the Wikipedia sentences:

# Partitioned the code into multiple chunks for your convenience
```

```
# Extract these entity pairs for all the sentences in our data:
```

Try: a) Implement the code to identify words as nouns, verbs, adjectives, adverbs, etc.
b) Use part of speech tagging to mark a word to its part of speech tag based on its context in the data. It is also used to extract relationships between words.
c) On top of Part of Speech tagging. It groups word into "chunks", mainly of noun phrases.
Chunking is done using regular expressions.

14. Transformers

14.1 Language Translation with Transformers

Natural Language Processing (NLP) is a field at the convergence of artificial intelligence, and linguistics. The aim of this exercise is to make the computers understand real-world language or natural language so that they can perform tasks like Question Answering, Language Translation, and many more.

Input: Text of any language (Multi30k dataset)

Output: Text of the desired language.

Hint:

```
# Download the dataset and also tokenize a raw text
import math
import torchtext
import torch
import torch.nn as nn
from torchtext.data.utils import
get_tokenizer from collections import
Counter
from torchtext.vocab import Vocab
from torchtext.utils import download_from_url,
extract_archive from torch.nn.utils.rnn import pad_sequence
from torch.utils.data import DataLoader
from torch import Tensor
from torch.nn import (TransformerEncoder,
TransformerDecoder,TransformerEncoderLayer,
TransformerDecoderLayer) import io
import time

url_base =
'https://raw.githubusercontent.com/multi30k/dataset/master/data/task1/raw/'
train_urls = ('train.de.gz', 'train.en.gz')
val_urls = ('val.de.gz', 'val.en.gz')
test_urls = ('test_2016_flickr.de.gz', 'test_2016_flickr.en.gz')

train_filepaths = [extract_archive(download_from_url(url_base + url))[0] for url
in train_urls]
val_filepaths = [extract_archive(download_from_url(url_base + url))[0] for url in
val_urls]
test_filepaths = [extract_archive(download_from_url(url_base + url))[0] for url in
test_urls]

de_tokenizer = get_tokenizer('spacy', language='de_core_news_sm')
en_tokenizer = get_tokenizer('spacy', language='en_core_web_sm')
```

```
def build_vocab(filepath, tokenizer):
counter = Counter()
with io.open(filepath, encoding="utf8") as f:
for string_ in f:
counter.update(tokenizer(string_))
return Vocab(counter, specials=['<unk>', '<pad>', '<bos>', '<eos>'])

de_vocab = build_vocab(train_filepaths[0], de_tokenizer)
en_vocab = build_vocab(train_filepaths[1], en_tokenizer)

def data_process(filepaths):
raw_de_iter        =        iter(io.open(filepaths[0],
encoding="utf8"))             raw_en_iter             =
iter(io.open(filepaths[1], encoding="utf8")) data = []
for (raw_de, raw_en) in zip(raw_de_iter, raw_en_iter):
de_tensor_ = torch.tensor([de_vocab[token] for token in
de_tokenizer(raw_de.rstrip("n"))],
dtype=torch.long)
en_tensor_ = torch.tensor([en_vocab[token] for token in
en_tokenizer(raw_en.rstrip("n"))],
dtype=torch.long)
data.append((de_tensor_, en_tensor_))
return data

train_data =
data_process(train_filepaths) val_data =
data_process(val_filepaths) test_data =
data_process(test_filepaths)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')


BATCH_SIZE = 128
PAD_IDX                =
de_vocab['<pad>']
BOS_IDX                =
de_vocab['<bos>']
EOS_IDX                =
de_vocab['<eos>']

# Write the code to customize the loading order and memory pinning by using dataloader


# Write the code to implement the transformer

# Represent the token embedding and get the notions of word

order. # Create the subsequent word mask and stop the target

word
# define the model parameters and instantiate the model.

# Define two different functions, that is for train and

evaluation. # Train and evaluate the model
```


Try: Implement the code to perform text translation and compare it with google translator.

15. Final Notes

The only way to learn programming is program, program, and program on challenging problems. The problems in this tutorial are certainly NOT challenging. There are tens of thousands of challenging problems available – used in training for various programming contests. Check out these sites:

- Sentiment Analysis on Movie Reviews: https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews
- Quora Question Pairs: https://www.kaggle.com/c/quora-question-pairs
- Toxic Comment Classification Challenge: https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge
- Natural Language Processing with Disaster Tweets: https://www.kaggle.com/c/nlp-getting-started
- Jigsaw Multilingual Toxic Comment Classification: https://www.kaggle.com/c/jigsaw-multilingual-toxic-comment-classification
- Google QUEST Q&A Labeling: https://www.kaggle.com/c/google-quest-challenge
- Text Normalization Challenge: https://www.kaggle.com/c/text-normalization-challenge-english-language
- Natural Language Processing 2: https://www.kaggle.com/c/jigsaw-unintended-bias-in-toxicity-classification

Other Competitions:

- U.S. Patent Phrase to Phrase Matching
- Feedback Prize — Predicting Effective Arguments
- NBME — Score Clinical Patient Notes
- Feedback Prize — Evaluating Student Writing
- Jigsaw Rate Severity of Toxic Comments
- Jigsaw Multilingual Toxic Comment Classification
- Jigsaw Unintended Bias in Toxicity Classification
- Toxic Comment Classification Challenge
- chaii — Hindi and Tamil Question Answering
- CommonLit Readability Prize
- Coleridge Initiative — Show US the Data
- Tweet Sentiment Extraction
- Google QUEST Q&A Labeling
- TensorFlow 2.0 Question Answering
- Gendered Pronoun Resolution
- Quora Insincere Questions Classification
- Avito Demand Prediction Challenge
- Text Normalization Challenge — Russian Language
- Text Normalization Challenge — English Language
- Personalized Medicine: Redefining Cancer Treatment
- Quora Question Pairs
- Two Sigma Connect: Rental Listing Inquiries
- The Allen AI Science Challenge
- Microsoft Malware Classification Challenge (BIG 2015)
- Tradeshift Text Classification
- Large Scale Hierarchical Text Classification

- StumbleUpon Evergreen Classification Challenge
- Detecting Insults in Social Commentary
- Predict Closed Questions on Stack Overflow
- The Hewlett Foundation: Automated Essay Scoring

Student must have any one of the following certifications:

1. NPTEL – Natural Language Processing
2. NOC: Applied Natural Language Processing
3. Natural Language Processing with Deep Learning – Stanford Online

V. REFERENCE BOOKS:

1. Christopher D. Manning and Hinrich Schutze, "Foundations of Natural Language Processing", The MIT Press Cambridge, Massachusetts London, England, 6th edition, 2003.
2. Daniel Jurafsky and James H. Martin "Speech and Language Processing", Prentice Hall, 3rd edition 2009.

VI. WEB REFERENCES:

1. https://www.geeksforgeeks.org/natural-language-processing-overview/
2. https://www.geeksforgeeks.org/python-word-similarity-using-spacy/?ref=rp
3. https://pub.towardsai.net/natural-language-processing-nlp-with-python-tutorial-for-beginners-1f54e610a1a0
4. https://www.analyticsvidhya.com/blog/2021/02/basics-of-natural-language-processing-nlp-basics/
5. https://towardsdatascience.com/free-hands-on-tutorials-to-get-started-in-natural-language-processing6a378e24dbfc.

VII. MATERIAL ONLINE:

1. Course Template
2. Lab Manual