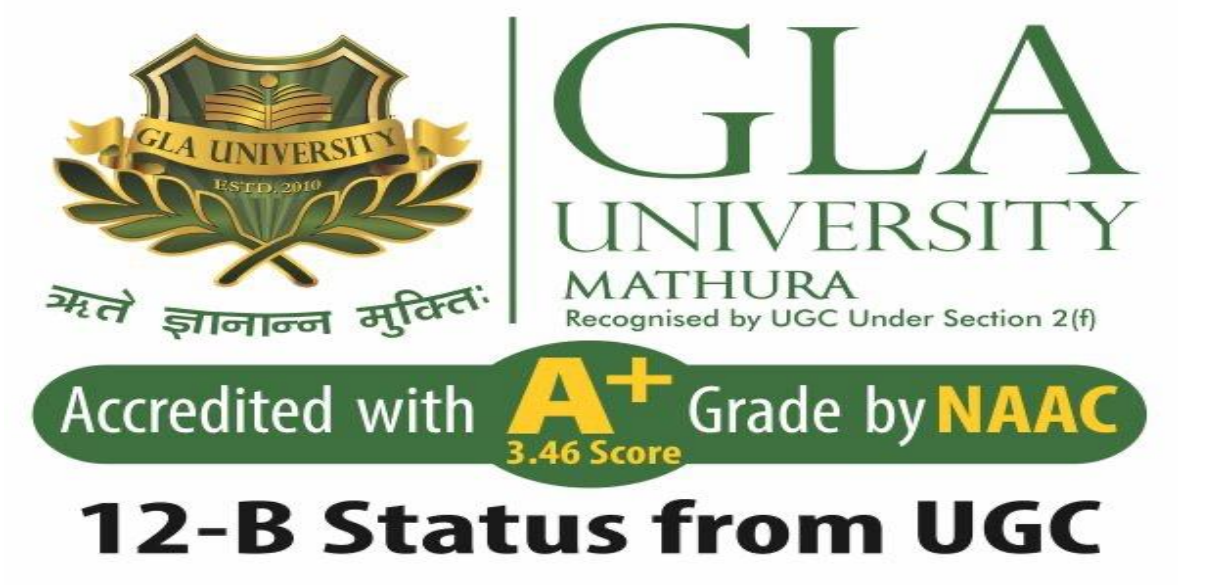
NATURAL LANGUAGE PROCESSING

LAB MANUAL



DEPARTMENT OF COMPUTER ENGINEERING AND APPLICATIONS

(CSE-AIML)

GLA UNIVERSITY

Lab Objectives:

1. Be able to discuss the current and likely future performance of several NLP applications.
2. Be able to describe briefly a fundamental technique for processing language for several subtasks, such as morphological processing.
3. Implement parsing, word sense disambiguation and etc.
4. Understand how these techniques draw on and relate to other areas of computer science .
5. Understand the basic principles of designing and running an NLP experiment.

Lab Outcomes:

Upon successful completion of this course, the students will be able to:

1. Student will be able to implement LSI, NER.
2. Student will be able to implement TD-IDF method and Ngram models
3. Develop a Part of speech tagger.
4. Student can able classify the text based on part of speech tagger.
5. Student can able to implement several NLP applications.

INDEX

S.No.	Week	Program Name
1	Week1	**a.Write a python program to perform tokenization by word and sentence using nltk.**
		b.Write a python program to eliminate stopwords using nltk.
		c.Write a python program to perform stemming using nltk.
2	Week2	**a.Write a python program to perform Parts of Speech tagging using nltk.**
		b.Write a python program to perform lemmatization using nltk.
3	Week3	**a.Write a python program for chunking using nltk.**
		b.Write a python program to perform Named Entity Recognition using nltk.
4	Week4	**a.Write a python program to find Term Frequency and Inverse Document Frequency (TF-IDF).**
		b.Write a python program for CYK parsing (Cocke- Younger-Kasami Parsing) or Chart Parsing.
5	Week5	**a. Write a python program to find all unigrams, bigrams and trigrams present in the given corpus.**
		b.Write a python program to find the probability of the given statement "This is my cat" by taking the an exmple corpus into consideration.
6	Week6	**Use the Stanford named Entity recognizer to extract entities from the documents. Use it programmatically and output for each document which named entities it contains and of which type.**
		Choose any corpus available on the internet freely. For the corpus, for each document, count how many times each stop word occurs and find out

7	Week7	**which are the most frequently occurring stop words. Further, calculate the term frequency and inverse document frequency as The motivation behind this is basically to find out how important a document is to a given query. For e.g.: If the query is say: "The brown crow". "The" is less important. "Brown" and "crow" are relatively more important. Since "the" is a more common word, its tf will be high. Hence we multiply it by idf, by knowing how common it is to reduce its weight.**
8	**Week8**	**Write the python code to perform sentiment analysis using NLP**
9	**Week9**	**Write the python code to develop Spam Filter using NLP**
10	**Week10**	**Write the python code to detect Fake News using NLP**

WEEK-1 Date:

Aim: a) Write a python program to perform tokenization by word and sentence using nltk. Program for sentence tokenization:

```
import nltk
nltk.download('punkt')  # Download the necessary tokenization models

from nltk.tokenize import sent_tokenize

def tokenize_sentences(text):
    sentences = sent_tokenize(text)
    return sentences

# Example text
text = "NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum."

# Tokenize sentences
sentences = tokenize_sentences(text)

# Print tokenized sentences
for i, sentence in enumerate(sentences):
    print(f"Sentence {i+1}: {sentence}")
```

Output:

Program for word Tokenization:

```
import nltk
nltk.download('punkt')  # Download the necessary tokenization models

from nltk.tokenize import word_tokenize

def tokenize_words(text):
        words = word_tokenize(text)
        return words

# Example text
text = "NLTK is a leading platform for building Python programs to work with human language data."

# Tokenize words
words = tokenize_words(text)

# Print tokenized words
print(words)
```

Output:

b.Write a python program to eliminate stopwords using nltk.

```
# Stopwords
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Download NLTK stopwords and tokenizer models
nltk.download('stopwords')
nltk.download('punkt')

def remove_stopwords(text):
    # Tokenize the text into words
    words = word_tokenize(text)

    # Get English stopwords
    english_stopwords = set(stopwords.words('english'))

    # Remove stopwords from the tokenized words
    filtered_words = [word for word in words if word.lower() not in english_stopwords]

    # Join the filtered words back into a single string
    filtered_text = ' '.join(filtered_words)

    return filtered_text

# Example text
text = "NLTK is a leading platform for building Python programs to work with human language data."

# Remove stopwords
filtered_text = remove_stopwords(text)

# Print filtered text
print(filtered_text)
```

Output:

c.Write a python program to perform stemming using nltk.

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

# Download NLTK tokenizer and stemmer models
nltk.download('punkt')

def stem_text(text):
   # Initialize the Porter Stemmer
   porter_stemmer =
   PorterStemmer() # Tokenize the
   text into words words =
   word_tokenize(text)
   # Apply stemming to each word
   stemmed_words = [porter_stemmer.stem(word) for word in
   words] # Join the stemmed words back into a single string
   stemmed_text = ' '.join(stemmed_words)
   return stemmed_text

# Example text
text = "NLTK is a leading platform for building Python programs to work with human language data."

# Perform stemming
stemmed_text =
stem_text(text)

#  Print  stemmed
text
print(stemmed_tex
t)
```

Output:

EXERCISE:

1. Write a python program to perform tokenization by word and sentence using Stanza.

2. Write a python program for word tokenization and sentence segmentation using spaCy.
   Write a python program to find all the stopwords in the given corpus using spaCy

WEEK-2 Date:

a. Write a python program to perform Parts of Speech tagging using nltk.

Parts of Speech Tagging

```
import nltk
from nltk.tokenize import word_tokenize

# Download NLTK tokenizer and POS tagging models
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')

def pos_tagging(text):
    # Tokenize the text into words
    words = word_tokenize(text)

    # Perform POS tagging
    tagged_words = nltk.pos_tag(words)

    return tagged_words

# Example text
text = "NLTK is a leading platform for building Python programs to work with human language data."

# Perform POS tagging
tagged_text = pos_tagging(text)

# Print POS tagged text
print(tagged_text)
```

Output:

b. Write a python program to perform lemmatization using nltk.

```
#Lemmatization
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

nltk.download('punkt')
nltk.download('wordnet')

def lemmatize_text(text):
    lemmatizer = WordNetLemmatizer()
    tokens = word_tokenize(text)
    lemmatized_text = ' '.join([lemmatizer.lemmatize(word) for word in tokens])
    return lemmatized_text

text = "The cats are chasing mice and playing in the garden"
lemmatized_text = lemmatize_text(text)
print("Original Text:", text)
print("Lemmatized Text:", lemmatized_text)
```

Output:

Signature of the Faculty

EXERCISE:

1. **Study and use the Stanford part of speech tagger on a suitable corpus available freely. The corpus should be decent size.(Use spaCy and stanza)**

2. **Write a python program for lemmatization using spaCy and stanza.**

WEEK-3 Date:

a. Write a python program for chunking using nltk.

```
#Chunking
import nltk
from nltk.tokenize import word_tokenize
from nltk import pos_tag, RegexpParser


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

def chunk_sentence(sentence):
    words = word_tokenize(sentence)
    tagged_words = pos_tag(words)

    # Define grammar for chunking
    grammar = r"""
    NP: {<DT|JJ|NN.*>+}          # Chunk sequences of DT, JJ, NN
    PP: {<IN><NP>}               # Chunk prepositions followed by NP
    VP: {<VB.*><NP|PP|CLAUSE>+$}  # Chunk verbs and their arguments
    CLAUSE: {<NP><VP>}           # Chunk NP, VP pairs
    """
    parser = RegexpParser(grammar)
    chunked_sentence = parser.parse(tagged_words)

    return chunked_sentence

sentence = "The quick brown fox jumps over the lazy dog"
chunked_sentence = chunk_sentence(sentence)
print(chunked_sentence)
```

Output:

b. Write a python program to perform Named Entity Recognition using nltk.

```
#Named Entity Recognition
import nltk
from nltk.tokenize import word_tokenize
from nltk import pos_tag, ne_chunk

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

def ner(text):
    words = word_tokenize(text)
    tagged_words = pos_tag(words)
    named_entities = ne_chunk(tagged_words)
    return named_entities

text = "Apple is a company based in California, United States. Steve Jobs was one of its founders."
named_entities = ner(text)
print(named_entities)
```

Output:

Signature of the faculty

EXERCISE:

1. Write a python program for chinking using nltk.

2. Use the Stanford named Entity recognizer to extract entities from the documents. Use it programmatically andoutput for each document which named entities it contains and of which type.

WEEK-4 **Date:**

a. Write a python program to find Term Frequency and Inverse Document Frequency (TF-IDF).

```
#tf-idf
import nltk
import string
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer

nltk.download('punkt')# Sample documents
documents = [
    "This is the first document.",
    "This document is the second document.",
    "And this is the third one.",
    "Is this the first document?",
]

# Tokenize and preprocess the documents
def preprocess_text(doc):
    # Tokenize the document into words
    tokens = nltk.word_tokenize(doc)

    # Remove punctuation
    tokens = [word for word in tokens if word not in string.punctuation]

    # Convert words to lowercase
    tokens = [word.lower() for word in tokens]

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words]

    # Join the tokens back into a single string
    preprocessed_doc = ' '.join(tokens)

    return preprocessed_doc

# Preprocess all documents
preprocessed_documents = [preprocess_text(doc) for doc in documents]

# Compute TF-IDF scores using scikit-learn
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(preprocessed_documents)

# Print TF-IDF matrix
print(tfidf_matrix.toarray())
```

Output:

Write a python program for CYK parsing (Cocke-Younger-Kasami Parsing) or Chart Parsing.

```
import nltk
grammar = nltk.CFG.fromstring(""" S -> V NP
V -> 'describe' | 'present' NP -> PRP N
PRP -> 'your' N -> 'work'
""")
parser = nltk.ChartParser(grammar) sent = 'describe your work'.split() print (list(parser.parse(sent)))
```

Output:

EXERCISE:

1. **Write a python program for CYK Parsing by defining your own Grammar.**

WEEK-5 Date:

a. **Write a python program to find all unigrams, bigrams and trigrams present in the given corpus.**

```
import nltk nltk.download('punkt')
from nltk.util import ngrams
samplText='this is a very good book to study' for i in range(1,4):
NGRAMS=ngrams(sequence=nltk.word_tokenize(samplText), n=i) for grams in NGRAMS:
print(grams)
```

b. Write a python program to find the probability of the given statement "This is my cat" by taking the an exmple corpus into consideration.

'This is a dog', 'This is a cat', 'I love my cat', 'This is my name'

```
def readData():
    data = ['This is a dog','This is a cat','I love my cat','This is my name ']
    dat=[]
    for i in range(len(data)):
        for word in data[i].split():
            dat.append(word)
    print(dat)
    return dat

def createBigram(data):
    listOfBigrams = []
    bigramCounts = {}
    unigramCounts = {}
    for i in range(len(data)-1):
        if i < len(data) - 1 and data[i+1].islower():

            listOfBigrams.append((data[i], data[i + 1]))

            if (data[i], data[i+1]) in bigramCounts:
                bigramCounts[(data[i], data[i + 1])] += 1
            else:
                bigramCounts[(data[i], data[i + 1])] = 1

        if data[i] in unigramCounts:
            unigramCounts[data[i]] += 1
        else:
            unigramCounts[data[i]] = 1
```

```
    return listOfBigrams, unigramCounts, bigramCounts


def calcBigramProb(listOfBigrams, unigramCounts, bigramCounts):
    listOfProb = {}
    for bigram in listOfBigrams:
        word1 = bigram[0]
        word2 = bigram[1]
        listOfProb[bigram] = (bigramCounts.get(bigram))/(unigramCounts.get(word1))
    return listOfProb


if__name__== '_main_':
    data = readData()
    listOfBigrams, unigramCounts, bigramCounts = createBigram(data)

    print("\n All the possible Bigrams are ")
    print(listOfBigrams)

    print("\n Bigrams along with their frequency ")
    print(bigramCounts)

    print("\n Unigrams along with their frequency ")
    print(unigramCounts)

    bigramProb = calcBigramProb(listOfBigrams, unigramCounts, bigramCounts)

    print("\n Bigrams along with their probability ")
    print(bigramProb)
    inputList="This is my cat"
    splt=inputList.split()
    outputProb1 = 1
    bilist=[]
    bigrm=[]
```

```
for i in range(len(splt) - 1):
    if i < len(splt) - 1:

        bilist.append((splt[i], splt[i + 1]))

print("\n The bigrams in given sentence are ")
print(bilist)
for i in range(len(bilist)):
    if bilist[i] in bigramProb:

        outputProb1 *= bigramProb[bilist[i]]
    else:

        outputProb1 *= 0
print('\n' + 'Probablility of sentence \"This is my cat\" = ' + str(outputProb1))
```

Signature of the Faculty

WEEK– 6 Date:

Use the Stanford named Entity recognizer to extract entities from the documents. Use it programmatically and output for each document which named entities it contains and of Which type.

Signature of the Faculty

WEEK– 7 Date:

Choose any corpus available on the internet freely. For the corpus, for each document, count how many times each stop word occurs and find out which are the most frequently occurring stop words. Further, calculate the term frequency and inverse document frequency as The motivation behind this is basically to find out how important a document is to a given query. For e.g.: If the query is say: “The brown crow”. “The” is less important. “Brown” and “crow” are relatively more important. Since “the” is a more common word, its tf will be high. Hence we multiply it by idf, by knowing how common it is to reduce its weight.

Signature of the Faculty

WeeK- 8 **Date:**

a. Write the python code to perform sentiment analysis using NLP

Signature of the Faculty

WEEK– 9 Date:

1. Write the python code to develop Spam Filter using NLP

Signature of the Faculty

Week-10:

Date:

1. Write the python code to detect Fake News using NLP

Signature of the Faculty