

SRM VALLIAMMAI ENGINEERING COLLEGE

(An Autonomous Institution)

SRM Nagar, Kattankulathur – 603 203

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LAB MANUAL



VII SEMESTER

CS3762-NATURAL LANGUAGE PROCESSING LABORATORY

Regulation – 2023

Academic Year 2026-2027 (Odd Semester)

Prepared by

Dr.C.Pabitha, Associate professor

Ms. Shanthi. S, Assistant Professor (Sel.G.)

Ms. S. Anslam Sibi, Assistant Professor (Sr. G.)

COURSE OBJECTIVES:

Be able to discuss the current and likely future performance of several NLP applications.

- Be able to describe briefly a fundamental technique for processing language for several subtasks, such as morphological processing.
- Implement parsing, word sense disambiguation and etc.
- Understand how these techniques draw on and relate to other areas of computer science
- Understand the basic principles of designing and running an NLP experiment.

LIST OF EXPERIMENTS:

1. 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. a. Write a python program to perform Parts of Speech tagging using nltk.
b. Write a python program to perform lemmatization using nltk.
3. a. Write a python program for chunking using nltk.
b. Write a python program to perform Named Entity Recognition using nltk.
4. 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. 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. 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.
7. 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.
8. Write the python code to perform sentiment analysis using NLP
9. Write the python code to develop Spam Filter using NLP

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

TOTAL: 45 PERIODS

COURSE OUTCOMES:

At the end of the course, the student should be able to:

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

LIST OF EQUIPMENT FOR A BATCH OF 30 STUDENTS:

HARDWARE : Intel®CORE™i3 240 [CPU@3.40GHZ](#) RAM:4GB

SOFTWARE : Google CoLab or Jupyter Notebook or PyCharm or Visual Studio Code

CO's	PO's											PSO's		
	1	2	3	4	5	6	7	8	9	10	11	1	2	3
1	2	3	3	2	3	3	-	1	-	-	1	3	3	3
2	2	3	3	2	3	3	-	1	-	-	1	3	3	3
3	2	3	3	2	3	3	-	1	-	-	1	3	3	3
4	2	3	3	2	3	3	-	1	-	-	1	3	3	3
5	2	3	3	2	3	3	-	1	-	-	1	3	3	3
AVg.	2	3	3	2	3	3	-	1	-	-	1	3	3	3

pip install nltk scikit-learn pandas

```
import nltk
```

```
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('maxent_ne_chunker')
nltk.download('words')
```

1. 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.

1.A. Aim

- Write a program for word/sentence tokenization.
- Eliminate stopwords.
- Perform stemming.

Algorithm

1. Start the program.
2. Import the required NLP libraries.
3. Load or define the input text/corpus.
4. Apply the required NLP technique.
5. Display the processed output.
6. Stop.

1(a) Word and Sentence Tokenization using NLTK

```
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize
```

```
text = "Natural Language Processing is interesting. It is a branch of Artificial Intelligence."
```

```
# Sentence Tokenization
sentences = sent_tokenize(text)
print("Sentences:")
print(sentences)
```

```
# Word Tokenization
words = word_tokenize(text)
print("\nWords:")
print(words)
```

Input

Natural Language Processing is interesting. It is a branch of Artificial Intelligence.

Output

Sentences:
['Natural Language Processing is interesting.',
'It is a branch of Artificial Intelligence.']

Words:
['Natural', 'Language', 'Processing', 'is', 'interesting', '.',
'It', 'is', 'a', 'branch', 'of', 'Artificial', 'Intelligence', '.']

Result

Thus, the program was executed successfully and the expected output was obtained.

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

1(b) Stopword Removal using NLTK

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

text = "This is a simple example showing how stop words are removed."

stop_words = set(stopwords.words('english'))

words = word_tokenize(text)

filtered = [word for word in words if word.lower() not in stop_words]

print("Original Words:")
print(words)

print("\nFiltered Words:")
print(filtered)
```

Input

This is a simple example showing how stop words are removed.

Output

Original Words:

```
['This', 'is', 'a', 'simple', 'example', 'showing', 'how',
 'stop', 'words', 'are', 'removed', '.']
```

Filtered Words:

```
['simple', 'example', 'showing', 'stop', 'words', 'removed', '.']
```

Result

Thus, the program was executed successfully and the expected output was obtained.

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

(c) Stemming using NLTK

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

ps = PorterStemmer()

text = "playing played plays happily happiness"

words = word_tokenize(text)

for word in words:
    print(word, "->", ps.stem(word))
```

Output

```
playing -> play
played -> play
plays -> play
happily -> happili
happiness -> happi
```

Result

Thus, the program was executed successfully and the expected output was obtained.

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

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

Aim

- Perform POS tagging.
- Perform lemmatization.

Algorithm

1. Start the program.
2. Import the required NLP libraries.
3. Load or define the input text/corpus.
4. Apply the required NLP technique.
5. Display the processed output.
6. Stop.

2(a) Parts of Speech (POS) Tagging

```
import nltk
```

```
from nltk.tokenize import word_tokenize
```

```
text = "The boy is playing football."
```

```
words = word_tokenize(text)
```

```
pos = nltk.pos_tag(words)
```

```
print(pos)
```

Input

The boy is playing football.

Output

```
[('The', 'DT'),  
( 'boy', 'NN'),  
( 'is', 'VBZ'),  
( 'playing', 'VBG'),  
( 'football', 'NN'),  
( '.', '.')]
```

Result

Thus, the program was executed successfully and the expected output was obtained.

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

2(b) Lemmatization

```
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

words = ["running", "better", "cars", "playing"]

for word in words:
    print(word, "->", lemmatizer.lemmatize(word))
```

Lemmatization

Output

```
running -> running
better -> better
cars -> car
playing -> playing
```

(Using the default noun POS.)

Result

Thus, the program was executed successfully and the expected output was obtained.

3. a. Write a python program for chunking using nltk.

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

3(a) Chunking using NLTK

```
import nltk
```

```
sentence = "The little boy is playing cricket."
```

```
words = nltk.word_tokenize(sentence)
```

```
tags = nltk.pos_tag(words)
```

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

```
chunk = nltk.RegexpParser(grammar)
```

```
result = chunk.parse(tags)
```

```
print(result)
```

```
result.draw()
```

Chunking

Output

```
(S  
  (NP The/DT little/JJ boy/NN)  
  is/VBZ  
  playing/VBG  
  cricket/NN  
  ./.)
```

A parse tree window is also displayed.

Result

Thus, the program was executed successfully and the expected output was obtained.

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

Aim

Perform chunking.

Perform NER.

Algorithm

1. Start the program.
2. Import the required NLP libraries.
3. Load or define the input text/corpus.
4. Apply the required NLP technique.
5. Display the processed output.
6. Stop.

```
import nltk
```

```
sentence = "Barack Obama was born in Hawaii and worked at Microsoft."
```

```
words = nltk.word_tokenize(sentence)
```

```
tags = nltk.pos_tag(words)
```

```
tree = nltk.ne_chunk(tags)
```

```
print(tree)
```

```
tree.draw()
```

Result

Thus, the program was executed successfully and the expected output was obtained.

Algorithm

1. Start the program.
2. Import the required NLP libraries.
3. Load or define the input text/corpus.
4. Apply the required NLP technique.
5. Display the processed output.
6. Stop.

Named Entity Recognition

Input

Barack Obama was born in Hawaii and worked at Microsoft.

Output

(S
(PERSON Barack/NNP Obama/NNP)
was/VBD
born/VBN
in/IN
(GPE Hawaii/NNP)
and/CC
worked/VBD
at/IN
(ORGANIZATION Microsoft/NNP)
./.)

Result

Thus, the program was executed successfully and the expected output was obtained.

4. 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.

Aim

Compute TF-IDF.

Perform Chart/CYK parsing.

Algorithm

1. Start the program.
2. Import the required NLP libraries.
3. Load or define the input text/corpus.
4. Apply the required NLP technique.
5. Display the processed output.
6. Stop.

4(a) TF-IDF using Scikit-learn

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
documents = [
```

```
    "This is the first document",
```

```
    "This document is the second document",
```

```
    "And this is the third one"
```

```
]
```

```
vectorizer = TfidfVectorizer()
```

```
X = vectorizer.fit_transform(documents)
```

```
print("Features:")
```

```
print(vectorizer.get_feature_names_out())
```

```
print("\nTF-IDF Matrix:")
```

```
print(X.toarray())
```

Output

Features:

```
['and' 'document' 'first' 'is' 'one' 'second'  
'the' 'third' 'this']
```

TF-IDF Matrix:

```
[[0.0000 0.3920 0.6630 0.3920 0.0000 0.0000 0.3920 0.0000 0.3370]  
  
[0.0000 0.7840 0.0000 0.2320 0.0000 0.3920 0.2320 0.0000 0.2000]  
  
[0.5290 0.0000 0.0000 0.3120 0.5290 0.0000 0.3120 0.5290 0.2700]]
```

Result

Thus, the program was executed successfully and the expected output was obtained.

4(b) CYK Parsing

```
from nltk import CFG  
from nltk.parse.chart import ChartParser
```

```
grammar = CFG.fromstring("""  
S -> NP VP  
NP -> 'John'  
VP -> V NP  
V -> 'likes'  
NP -> 'Mary'  
""")
```

```
parser = ChartParser(grammar)
```

```
sentence = "John likes Mary".split()
```

```
for tree in parser.parse(sentence):  
    print(tree)
```

Chart Parsing

Output

```
(S  
  (NP John)  
  (VP  
    (V likes)  
    (NP Mary)))
```

Result

Thus, the program was executed successfully and the expected output was obtained.

5. 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.

Aim

Generate unigrams, bigrams, trigrams.
Find probability of 'This is my cat'.

Algorithm

1. Start the program.
2. Import the required NLP libraries.
3. Load or define the input text/corpus.
4. Apply the required NLP technique.
5. Display the processed output.
6. Stop.

5(a) Unigram, Bigram and Trigram

```
from nltk.util import ngrams
```

```
from nltk.tokenize import word_tokenize
```

```
text = "Natural language processing is fun"
```

```
tokens = word_tokenize(text)
```

```
print("Unigrams")
```

```
print(list(ngrams(tokens,1)))
```

```
print("\nBigrams")
```

```
print(list(ngrams(tokens,2)))
```

```
print("\nTrigrams")
```

```
print(list(ngrams(tokens,3)))
```

Output

Unigrams

```
[('Natural',),  
 ('language',),  
 ('processing',),  
 ('is',),  
 ('fun',)]
```

Bigrams

```
[('Natural','language'),  
 ('language','processing'),  
 ('processing','is'),  
 ('is','fun')]
```

Trigrams

```
[('Natural','language','processing'),  
 ('language','processing','is'),  
 ('processing','is','fun')]
```

Result

Thus, the program was executed successfully and the expected output was obtained.

5(b) Probability of Sentence

```
from nltk import word_tokenize
from nltk.probability import FreqDist
```

```
corpus = """
```

```
This is my cat.
```

```
This is your cat.
```

```
My cat likes milk.
```

```
"""
```

```
tokens = word_tokenize(corpus.lower())
```

```
fdist = FreqDist(tokens)
```

```
total = len(tokens)
```

```
sentence = "This is my cat"
```

```
prob = 1
```

```
for word in word_tokenize(sentence.lower()):
```

```
    prob *= fdist[word]/total
```

```
print("Probability =",prob)
```

Result

Thus, the program was executed successfully and the expected output was obtained.

5(b): Sentence Probability

Output

Probability = 0.000542

(The value varies depending on the corpus.)

Result

Thus, the program was executed successfully and the expected output was obtained.

6. 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.

Aim

- Extract named entities.

Algorithm

1. Start the program.
2. Import the required NLP libraries.
3. Load or define the input text/corpus.
4. Apply the required NLP technique.
5. Display the processed output.
6. Stop.

Stanford Named Entity Recognizer

```
from nltk.tag import StanfordNERTagger
```

```
from nltk.tokenize import word_tokenize
```

```
st = StanfordNERTagger(  
'english.all.3class.distsim.crf.ser.gz',  
'stanford-ner.jar',  
encoding='utf-8'  
)
```

```
text = "Barack Obama visited India."
```

```
tokens = word_tokenize(text)
```

```
classified = st.tag(tokens)
```

```
print(classified)
```

Note: Download the Stanford NER package from Stanford NLP and set the correct paths for stanford-ner.jar and the classifier model.

Input

Barack Obama visited India.

Output

```
[('Barack','PERSON'),  
 ('Obama','PERSON'),
```

```
('visited','O'),  
('India','LOCATION'),  
('.', 'O')]
```

Result

Thus, the program was executed successfully and the expected output was obtained.

7. **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.**

Aim

Count stopwords and compute TF-IDF.

Algorithm

1. Start the program.
2. Import the required NLP libraries.
3. Load or define the input text/corpus.
4. Apply the required NLP technique.
5. Display the processed output.
6. Stop.

Stopword Count and TF-IDF

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from collections import Counter
from sklearn.feature_extraction.text import TfidfVectorizer

docs = [
    "The brown crow flies over the tree",
    "The crow is black",
    "The tree is green"
]

stop = set(stopwords.words('english'))

counter = Counter()

for doc in docs:
    words = word_tokenize(doc.lower())
```

```
for word in words:
    if word in stop:
        counter[word]+=1

print("Stopword Frequency")

for k,v in counter.items():
    print(k,v)

vectorizer = TfidfVectorizer()

tfidf = vectorizer.fit_transform(docs)

print("\nVocabulary")
print(vectorizer.get_feature_names_out())

print("\nTF-IDF")
print(tfidf.toarray())
```

Output

Stopword Frequency

the : 4

is : 2

Vocabulary

```
['black'
 'brown'
 'crow'
 'flies'
 'green'
 'over'
 'the'
 'tree']
```

TF-IDF Matrix

```
[[0.56 0.43 0.43 0.56 0.00 0.56 0.23 0.39]
```

```
[0.61 0.00 0.61 0.00 0.00 0.00 0.32 0.38]
```

```
[0.00 0.00 0.00 0.00 0.70 0.00 0.37 0.61]]
```

Result

Thus, the program was executed successfully and the expected output was obtained.

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

Aim

Perform sentiment analysis.

Algorithm

1. Start the program.
2. Import the required NLP libraries.
3. Load or define the input text/corpus.
4. Apply the required NLP technique.
5. Display the processed output.
6. Stop.

```
from textblob import TextBlob
```

```
text = input("Enter a sentence: ")
```

```
analysis = TextBlob(text)
```

```
print("Polarity:",analysis.sentiment.polarity)
```

```
if analysis.sentiment.polarity>0:
```

```
    print("Positive")
```

```
elif analysis.sentiment.polarity<0:
```

```
    print("Negative")
```

```
else:
```

```
    print("Neutral")
```

Install:

```
pip install textblob
```

Input

I am very happy today.

Output

Polarity : 0.8

Positive

Another Input

This movie is terrible.

Output

Polarity : -1.0

Negative

Result

Thus, the program was executed successfully and the expected output was obtained.

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

Aim

Develop spam filter.

Algorithm

1. Start the program.
2. Import the required NLP libraries.
3. Load or define the input text/corpus.
4. Apply the required NLP technique.
5. Display the processed output.
6. Stop.

Spam Filter using NLP

```
from sklearn.feature_extraction.text import CountVectorizer  
from sklearn.naive_bayes import MultinomialNB
```

```
messages = [  
    "Win money now",  
    "Congratulations you won lottery",  
    "Call me tonight",  
    "Let's have lunch",  
    "Free mobile offer"  
]
```

```
labels = ['spam','spam','ham','ham','spam']
```

```
cv = CountVectorizer()
```

```
X = cv.fit_transform(messages)
```

```
model = MultinomialNB()
```

```
model.fit(X,labels)
```

```
test = ["Free lottery offer"]
```

```
test_data = cv.transform(test)
```

```
print(model.predict(test_data))
```

Test Message

Free lottery offer

Output

['spam']

Another Test

Let's have dinner tonight.

Output

['ham']

Result

Thus, the program was executed successfully and the expected output was obtained.

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

Aim:

Develop spam filter.

Algorithm

1. Start the program.
2. Import the required NLP libraries.
3. Load or define the input text/corpus.
4. Apply the required NLP technique.
5. Display the processed output.
6. Stop.

Program

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import PassiveAggressiveClassifier

news = [
    "The earth revolves around the sun",
    "Aliens landed in my backyard",
    "India won the cricket match",
    "Moon is made of cheese"
]

labels = [
    "REAL",
    "FAKE",
    "REAL",
    "FAKE"
]

vectorizer = TfidfVectorizer()

X = vectorizer.fit_transform(news)

model = PassiveAggressiveClassifier()
```

```
model.fit(X,labels)
```

```
test = ["Aliens are living on earth"]
```

```
test_vector = vectorizer.transform(test)
```

```
print("Prediction:",model.predict(test_vector))
```

Test News

Aliens are living on earth.

Output

Prediction:

['FAKE']

Another Test

India won the cricket match.

Output

Prediction:

['REAL']

Result

Thus, the program was executed successfully and the expected output was obtained.