INDEX

S.No	Experiments	Performed Date	Submission Date	Signature
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.			
2)	(a).Write a python program to perform pos 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. c. Implement re for regular expressions			
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 example 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 a program to perform Sentiment Analyis			
9)	Write a program to perform Text Claasification.			

0)	Write a python program to perform stemming using nltk.			

EXPERIMENT 1

Q.(a).Write a python program to perform tokenization by word and sentence using nltk.
(b). Write a python program to eliminate stopwords using nltk.

```
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
nltk.download('punkt')
nltk.download('stopwords')

data = "Natural Language Processing is a fascinating field."

# (a) : Word Tokenization
word_tokens = word_tokenize(data)
print("Word Tokens:", word_tokens)
sentence_tokens = sent_tokenize(data)
print("Sentence Tokens:", sentence_tokens)

# (b): Define stopwords list
stop_words = set(stopwords.words('english'))
filtered_words = [word for word in word_tokens if word.lower() not in stop_words]
print("Filtered Words (without stopwords):", filtered_words
```

OUTPUT

(a)

```
Word Tokens: ['Natural', 'Language', 'Processing', 'is', 'a', 'fascinating', 'field', '.']
Sentence Tokens: ['Natural Language Processing is a fascinating field.']
```

Filtered Words (without stopwords): ['Natural', 'Language', 'Processing', 'fascinating', 'field', '.']

EXPERIMENT 2

Q. (a).Write a python program to perform Parts of Speech tagging using nltk.
(b).Write a python program to perform lemmatization using nltk.

```
import nltk
from nltk.tokenize import word_tokenize
from nltk import pos_tag
from nltk.stem import WordNetLemmatizer

nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
nltk.download('omw-1.4')

data = "The foxes are running swiftly across the forest."
# (a): Parts of Speech Tagging
tokens = word_tokenize(data)
pos_tags = pos_tag(tokens)
print("Parts of Speech Tags:", pos_tags)

# Part b: Lemmatization
lemmatizer = WordNetLemmatizer()
lemmatized_words = [lemmatizer.lemmatize(word, pos='v') for word in tokens]
print("Lemmatized Words:", lemmatized_words)
```

OUTPUT

```
(a)
Parts of Speech Tags: [('The', 'DT'), ('foxes', 'NNS'), ('are', 'VBP'),
('running', 'VBG'), ('swiftly', 'RB'), ('across', 'IN'), ('the', 'DT'),
('forest', 'NN'), ('.', '.')]

(b)
Lemmatized Words: ['The', 'fox', 'be', 'run', 'swiftly', 'across', 'the', 'forest', '.']
```

EXPERIMENT 3

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

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

c. Implement re for regular expressions

```
import nltk
import re
from nltk import word_tokenize, pos_tag, ne_chunk

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

data = "John is working at Google in California. He joined the company in 2020."

# (a): Chunking
tokens = word_tokenize(data)
pos_tags = pos_tag(tokens)
chunk_grammar = "NP: {<DT>?<JJ>*<NN>}"  # Noun phrase: optional determiner,
adjectives, and noun
chunk_parser = nltk.RegexpParser(chunk_grammar)
chunk_tree = chunk_parser.parse(pos_tags)
print("Chunking Result:")
print(chunk_tree)
chunk_tree.draw()

# (b): Named Entity Recognition (NER)
ner_tree = ne_chunk(pos_tags)
print("Named Entity Recognition Result:")
print(ner_tree)
ner_tree.draw()

# (c): Regular Expressions
```

```
nple_text = "Contact us at support@example.com or visit https://example.com for more
info. Call us at +1-800-555-0199."
emails = re.findall(r'[\w.-]+@[\w.-]+', sample_text)
print("Extracted Emails:", emails)
urls = re.findall(r'https?://[\w./-]+', sample_text)
print("Extracted URLs:", urls)
phone_numbers = re.findall(r'\+?\d{1,2}-?\d{3}-\d{3}-\d{4}', sample_text)
print("Extracted Phone Numbers:", phone_numbers)
```

OUTPUT

(a)

Chunking Result:

```
(S
  John/NNP  is/VBZ  working/VBG  at/IN
  Google/NNP  in/IN  California/NNP  ./.
  He/PRP  joined/VBD
  (NP the/DT company/NN)
  in/IN  2020/CD
  ./.)
```

(b)

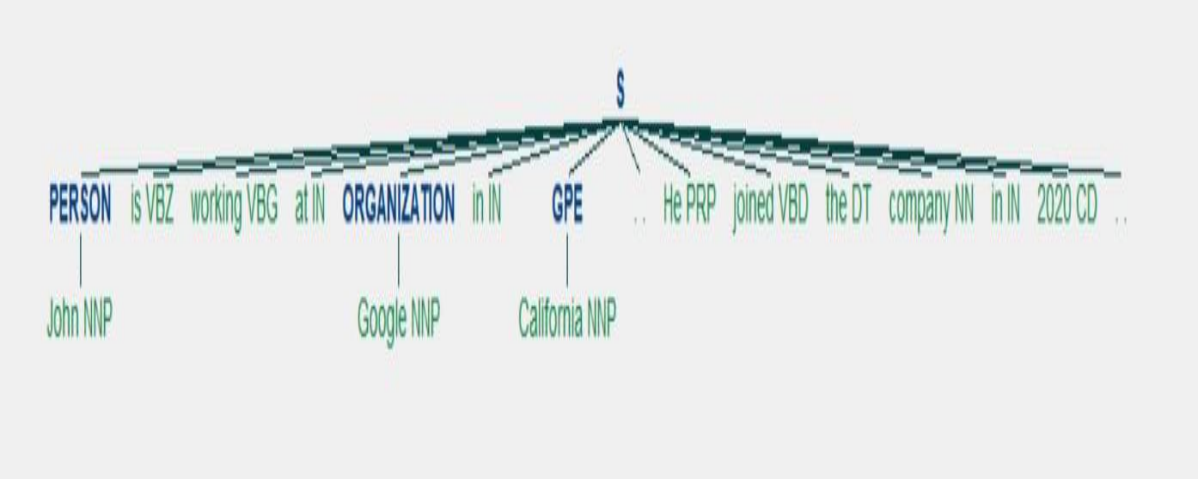
Named Entity Recognition Result:

```
(S
  (PERSON John/NNP)
  is/VBZ  working/VBG  at/IN
  (ORGANIZATION Google/NNP)
  in/IN  (GPE California/NNP)  ./.
  He/PRP  joined/VBD  the/DT  company/NN  in/IN  2020/CD
  ./.)
```

(c)

```
Output of Part c
Extracted Emails: ['support@example.com']
Extracted URLs: ['https://example.com']
```

Extracted Phone Numbers: ['+1-800-555-0199']



EXPERIMENT 4

Q. 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.

```
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
import nltk
from nltk import CFG

# (a): Term Frequency and Inverse Document Frequency (TF-IDF)
documents = [
    "Natural language processing makes human-machine interaction easier.",
    "Machine learning is a subset of artificial intelligence.",
    "Artificial intelligence and machine learning drive innovations."
]
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(documents)
print("TF-IDF Matrix:")
print(tfidf_matrix.toarray())
print("Feature Names:", vectorizer.get_feature_names_out())


# (b): CYK Parsing
cfg = CFG.fromstring("""
  S -> NP VP
  VP -> V NP | V
  NP -> Det N | Det N PP
  PP -> P NP
  Det -> 'the' | 'a'
  N -> 'man' | 'park' | 'dog' | 'telescope'
  V -> 'saw'
  P -> 'in' | 'with'
""")
```

```
sentence = "the man saw the dog with the telescope".split()
parser = nltk.ChartParser(cfg)
print("CYK Parsing Result:")
for tree in parser.parse(sentence):
    print(tree)
    tree.draw()
```

OUTPUT

(a)

TF-IDF Matrix:

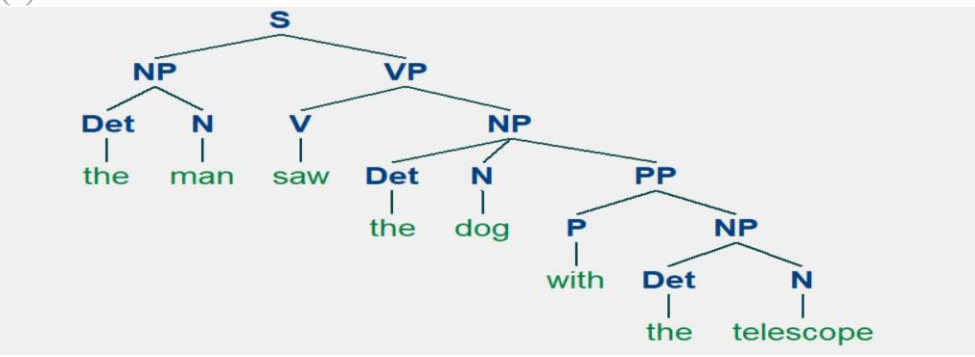
[[0. 0. 0. 0.36888498 0.36888498 0. 0. 0.36888498 0. 0.36888498 0. 0.21786941 0.36888498 0.36888498 0. 0.36888498 0. **]**

[0. 0.33729513 0. 0. 0. 0. 0.33729513 0. 0.44350256 0. 0.33729513 0.26193976 0. 0. 0.44350256 0. 0.44350256**]**

[0.44350256 0.33729513 0.44350256 0. 0. 0.44350256 0.33729513 0. 0. 0. 0.33729513 0.26193976 0. 0. 0. 0. 0. **]]**

Feature Names: ['and' 'artificial' 'drive' 'easier' 'human' 'innovations' 'intelligence' 'interaction' 'is' 'language' 'learning' 'machine' 'makes' 'natural' 'of' 'processing' 'subset']

(b)



EXPERIMENT 5

Q. (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 example corpus into consideration.

```
import nltk
from nltk import word_tokenize, ngrams, FreqDist
from collections import Counter
import math

nltk.download('punkt')

corpus = "This is my cat. This is my dog. My cat and dog are friends."

# (a):  Unigrams, Bigrams, and Trigrams
tokens = word_tokenize(corpus.lower())
unigrams = list(ngrams(tokens, 1))
bigrams = list(ngrams(tokens, 2))
trigrams = list(ngrams(tokens, 3))
print("Unigrams:", unigrams)
print("Bigrams:", bigrams)
print("Trigrams:", trigrams)


# (b): Probability of the statement "This is my cat"
statement = "This is my cat"
statement_tokens = word_tokenize(statement.lower())

bigram_fd = FreqDist(bigrams)
probability = 1

for i in range(len(statement_tokens) - 1):
    bigram = (statement_tokens[i], statement_tokens[i + 1])
    count_bigram = bigram_fd[bigram]
```

```
    count_word = tokens.count(statement_tokens[i])
    if count_word > 0:
        prob = count_bigram / count_word
    else:
        prob = 0
    probability *= prob

print(f"Probability of the statement '{statement}': {probability}")
```

OUTPUT

(a)

```
Unigrams: [('this',), ('is',), ('my',), ('cat',), ('.',), ('this',),
('is',), ('my',), ('dog',), ('.',), ('my',), ('cat',), ('and',),
('dog',), ('are',), ('friends',), ('.',)]
Bigrams: [('this', 'is'), ('is', 'my'), ('my', 'cat'), ('cat', '.'),
('.', 'this'), ('this', 'is'), ('is', 'my'), ('my', 'dog'), ('dog', '.'),
('.', 'my'), ('my', 'cat'), ('cat', 'and'), ('and', 'dog'), ('dog',
'are'), ('are', 'friends'), ('friends', '.')]
Trigrams: [('this', 'is', 'my'), ('is', 'my', 'cat'), ('my', 'cat', '.'),
('cat', '.', 'this'), ('.', 'this', 'is'), ('this', 'is', 'my'), ('is',
'my', 'dog'), ('my', 'dog', '.'), ('dog', '.', 'my'), ('.', 'my', 'cat'),
('my', 'cat', 'and'), ('cat', 'and', 'dog'), ('and', 'dog', 'are'),
('dog', 'are', 'friends'), ('are', 'friends', '.')]
```

(b)

```
Probability of the statement 'This is my cat': 0.6666666666666666
```

EXPERIMENT 6

Q. 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.

```
import os
from nltk.tag import StanfordNERTagger
from nltk.tokenize import word_tokenize

stanford_ner_jar = "path/to/stanford-ner.jar"
stanford_ner_model = "path/to/classifiers/english.all.3class.distsim.crf.ser.gz"
ner_tagger = StanfordNERTagger(
    model_filename=stanford_ner_model,
    path_to_jar=stanford_ner_jar
)
documents = [
    "Elon Musk founded SpaceX in 2002.",
    "Amazon is based in Seattle, Washington."
]
for i, doc in enumerate(documents):
    tokens = word_tokenize(doc)
    ner_tags = ner_tagger.tag(tokens)
    entities = {}
    for word, tag in ner_tags:
        if tag != "O":
            if tag not in entities:
                entities[tag] = []
            entities[tag].append(word)
    print(f"\nDocument {i + 1}:")
    print(f"Text: {doc}")
    print("Named Entities:")
    for entity_type, entity_list in entities.items():
        print(f"  {entity_type}: {', '.join(entity_list)}")
```

OUTPUT

Document 1:

Text: Elon Musk founded SpaceX in 2002.

Named Entities:

PERSON: Elon Musk

ORGANIZATION: SpaceX

Document 2:

Text: Amazon is based in Seattle, Washington.

Named Entities:

ORGANIZATION: Amazon

GPE: Seattle, Washington

EXPERIMENT 7

Q. 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.

```
import nltk
from nltk.corpus import stopwords
import math
from collections import defaultdict

nltk.download('stopwords')

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

def compute_tf(doc):
    tf = {}
    total_terms = len(doc)
    for word in doc:
        tf[word] = tf.get(word, 0) + 1
    for word in tf:
        tf[word] = tf[word] / total_terms
    return tf

def compute_idf(corpus):
    idf = {}
    total_docs = len(corpus)
    word_docs = defaultdict(int)

    for doc in corpus:
        unique_words = set(doc)
        for word in unique_words:
            word_docs[word] += 1
    for word, count in word_docs.items():
        idf[word] = math.log(total_docs / (1 + count))
    return idf

documents = [
    ["The", "cat", "sat", "on", "the", "mat"],
    ["The", "dog", "barked", "at", "the", "cat"]
]
```

```
stopword_counts = defaultdict(int)
filtered_documents = []

for doc in documents:
    filtered_doc = [word.lower() for word in doc if word.lower() not in stop_words]
    filtered_documents.append(filtered_doc)
    for word in doc:
        if word.lower() in stop_words:
            stopword_counts[word.lower()] += 1

sorted_stopwords = sorted(stopword_counts.items(), key=lambda x: x[1], reverse=True)

print("Most frequent stop words:")
for word, count in sorted_stopwords[:10]:
    print(f"{word}: {count}")

tf_values = [compute_tf(doc) for doc in filtered_documents]
idf_values = compute_idf(filtered_documents)

document_index = 0
tf_idf = {}
for word, tf_val in tf_values[document_index].items():
    idf_val = idf_values.get(word, 0)
    tf_idf[word] = tf_val * idf_val

print(f"\nTF-IDF for Document {document_index + 1}:")
for word, tfidf in sorted(tf_idf.items(), key=lambda x: x[1], reverse=True):
    print(f"{word}: {tfidf}")
```

OUTPUT

```
Most frequent stop words:
the: 4
at: 1
on: 1

TF-IDF for Document 1:
cat: 0.2876820724517809
mat: 0.2876820724517809
sat: 0.287682072451780
```

EXPERIMENT 8

Q.Write a program to perform Sentiment Analyis

```
pip install textblob
from textblob import TextBlob

from textblob import TextBlob

text_1 = "The movie was so awesome."
text_2 = "The food here tastes terrible."
text_3 ="I don't like this book."
text_4= " I love my country India"

#Determining the Polarity
p_1 = TextBlob(text_1).sentiment.polarity
p_2 = TextBlob(text_2).sentiment.polarity
p_3 = TextBlob(text_3).sentiment.polarity
p_4 = TextBlob(text_4).sentiment.polarity

#Determining the Subjectivity
s_1 = TextBlob(text_1).sentiment.subjectivity
s_2 = TextBlob(text_2).sentiment.subjectivity
s_3 = TextBlob(text_3).sentiment.subjectivity
s_4 = TextBlob(text_4).sentiment.subjectivity
print("Polarity of Text 1 is", p_1)
print("Polarity of Text 2 is", p_2)
print("Polarity of Text 3is", p_3)
print("Polarity of Text 4 is", p_4)
print("Subjectivity of Text 1 is", s_1)
print("Subjectivity of Text 2 is", s_2)
print("Subjectivity of Text 3 is", s_3)
print("Subjectivity of Text 4 is", s_4)
```

OUTPUT :

```
Polarity of Text 1 is 1.0
Polarity of Text 2 is -1.0
Polarity of Text 3is 0.0
Polarity of Text 4 is 0.5
Subjectivity of Text 1 is 1.0
Subjectivity of Text 2 is 1.0
Subjectivity of Text 3 is 0.0
Subjectivity of Text 4 is 0.6
```

EXPERIMENT 9

Q. Write a program to perform Text Claasification.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, accuracy_score

# Dataset Preparation
data = {
    "text": [
        "This is an amazing product",
        "I had a terrible experience with this",
        "Highly recommend this item",
        "Not worth the money",
        "Excellent quality and service",
        "Awful and disappointing"
    ],
    "label": ["Positive", "Negative", "Positive", "Negative", "Positive", "Negative"]
}
df = pd.DataFrame(data)

# Data Preprocessing
X = df['text']
y = df['label']

# Split Data into Train and Test Sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature Extraction using TF-IDF
vectorizer = TfidfVectorizer(stop_words='english', max_features=500)
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

# Model Training
model = MultinomialNB()
model.fit(X_train_tfidf, y_train)
```

```
# Model Prediction
y_pred = model.predict(X_test_tfidf)

# Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

OUTPUT:

	Precision	recall	f1-score	support
Negative	1.00	1.00	1.00	1
Positive	1.00	1.00	1.00	1
accuracy			1.00	2
macro avg	1.00	1.00	1.00	2

EXPERIMENT 10

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

```
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
nltk.download('punkt')
nltk.download('stopwords')

data = "Natural Language Processing is a fascinating field."
stemmer = PorterStemmer()
stemmed_words = [stemmer.stem(word) for word in word_tokens]
print("Stemmed Words:", stemmed_words)
```

OUTPUT:

```
Stemmed Words: ['natur', 'languag', 'process', 'is', 'a', 'fascin', 'field', '.']
```