

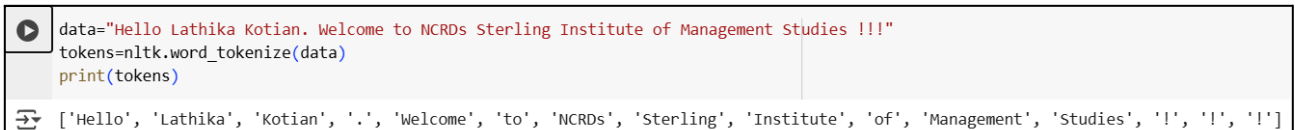
PRACTICAL - 1

Aim: Write a program to implement Tokenization of text

NLTK Code :

```
# 1.) TOKENIZATION using nltk
import nltk
nltk.download()
data="Hello Lathika Kotian. Welcome to NCRDs Sterling Institute of Management
Studies !!!"
tokens=nltk.word_tokenize(data)
print(tokens)
```

Output:



The screenshot shows a Jupyter Notebook cell with the following code and output:

```
data="Hello Lathika Kotian. Welcome to NCRDs Sterling Institute of Management Studies !!!"
tokens=nltk.word_tokenize(data)
print(tokens)
```

The output is a list of tokens: ['Hello', 'Lathika', 'Kotian', '.', 'Welcome', 'to', 'NCRDs', 'Sterling', 'Institute', 'of', 'Management', 'Studies', '!', '!', '!']

SpaCy Code:

```
# 1b.) TOKENIZATION using SpaCy
import spacy
nlp = spacy.load("en_core_web_sm")

data1 = "Hello Lathika Kotian. Welcome to NCRDs Sterling Institute of Management
Studies !!!"
doc = nlp(data1)

# Tokenize using spaCy
tokens = [token.text for token in doc]
print(tokens)
```

Output:

```
# 1b.) TOKENIZATION using SpaCy
import spacy
nlp = spacy.load("en_core_web_sm")

data1 = "Hello Lathika Kotian. Welcome to NCRDs Sterling Institute of Management Studies !!!"
doc = nlp(data1)

# Tokenize using spaCy
tokens = [token.text for token in doc]
print(tokens)
```

['Hello', 'Lathika', 'Kotian', '.', 'Welcome', 'to', 'NCRDs', 'Sterling', 'Institute', 'of', 'Management', 'Studies', '!', '!', '!']

PRACTICAL - 2

Aim: Write a program to implement Stop word removal.

NLTK Code:

```
print("PRACTICAL-2 NLTK(Stopword Removal) by Lathika Kotian - 98\n")
#2.) STOP WORD REMOVAL using nltk
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
nltk.download('stopwords')

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

print(f'Stopwords : \n{stopWords}')
data = "This is a simple example to demonstrate stop word removal in NLP. Success comes to those who work hard and stay focused on their goals"
tokens = nltk.word_tokenize(data)
filtered_Data = []
for w in tokens:
    if w not in stopWords:
        filtered_Data.append(w)
print("\nFiltered data after removing stopWords :\n", filtered_Data)
```

Output:

```
PRACTICAL-2 NLTK(Stopword Removal) by Lathika Kotian - 98

Stopwords :
{'but', 'ours', 'she'd', 'you'll', 'until', 'why', 'm', 'they', 'yours', 'weren', 'these', 'at', 'here', 've', 'we'd', 'more', 'wasn't', 'didn', 'an

Filtered data after removing stopWords :
['This', 'simple', 'example', 'demonstrate', 'stop', 'word', 'removal', 'NLP', '.', 'Success', 'comes', 'work', 'hard', 'stay', 'focused', 'goals']
```

Spacy Code:

```
print("PRACTICAL-2 SpaCy(Stopword Removal) by Lathika Kotian - 98\n")
```

#2.) STOP WORD REMOVAL using SpaCy

```
import spacy
nlp = spacy.load("en_core_web_sm") # Load the spaCy English model

data = "This is a simple example to demonstrate stop word removal in NLP. Success
comes to those who work hard and stay focused on their goals"
doc = nlp(data)# Process the text

# List of spaCy's stop words
stop_words = nlp.Defaults.stop_words
print(f"Stopwords:\n{stop_words}\n")

# Filter out stop words
filtered_data = [token.text for token in doc if not token.is_stop]

print("\nFiltered data after removing stopWords:\n",filtered_data)
```

Output:

```
PRACTICAL-2 SpaCy(Stopword Removal) by Lathika Kotian - 98

Stopwords:
{'ll', 'why', 're', 'my', 'also', 'no', 'before', 'have', 've', 'would', 'each', 'whereas', 'this', 'herein', 'might', 'along', 'used', 'a',
Filtered data after removing stopWords:
['simple', 'example', 'demonstrate', 'stop', 'word', 'removal', 'NLP', '.', 'Success', 'comes', 'work', 'hard', 'stay', 'focused', 'goals']
```

PRACTICAL - 3

Aim: Write a program to implement Stemming.

Code:

```
print("PRACTICAL-3 NLTK(PorterStemmer) by Lathika Kotian - 98\n")

import nltk
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
nltk.download('punkt')

port_stemmer = PorterStemmer()

default_sentence = "The children liked playing games, but they were quickly bored
and stopped liking them."

try:
    with open("PortStem.txt", "r") as file:
        text_data = file.read().strip()
        if not text_data:
            print("File is empty. Using default sentence.\n")
            text_data = default_sentence
except FileNotFoundError:
    print("File not found. Using default sentence.\n")
    text_data = default_sentence

# Tokenize and stem
tokens = word_tokenize(text_data)
print("Stemmed Words:")
for word in tokens:
    print("\t",word, ":", port_stemmer.stem(word))
```

Output:

PRACTICAL-3 NLTK(PorterStemmer) by Lathika Kotian - 98

Stemmed Words:

The : the
children : children
liked : like
playing : play
games : game
, : ,
but : but
they : they
were : were
quickly : quickli
bored : bore
and : and
stopped : stop
liking : like
them : them

PRACTICAL - 4

Aim: Write a program to implement Lemmatization

Spacy Code:

```
#4.) LEMMATIZATION using Spacy
print("PRACTICAL-4 spaCy (Lemmatizer) by Lathika Kotian - 98\n")

import spacy
nlp = spacy.load("en_core_web_sm")

words = ["socks", "corpora", "better"]
# Process each word and print its lemmatized form
for word in words:
    doc = nlp(word) # Process word through spaCy
    print(f"{word} :", doc[0].lemma_)
```

Output:

```
PRACTICAL-4 spaCy (Lemmatizer) by Lathika Kotian - 98

socks : sock
corpora : corpora
better : well
```

NLTK Code:

```
#4.) LEMMATIZATION using NLTK
print("PRACTICAL-4 NLTK (Lemmatization) by Lathika Kotian - 98\n")
from nltk.stem import WordNetLemmatizer
lemmati=WordNetLemmatizer()

print("Socks :",lemmati.lemmatize("socks"))
```

```
print("corpora :",lemmati.lemmatize("corpora"))  
print("better :",lemmati.lemmatize("better",pos="a"))
```

Output:

```
PRACTICAL-4 NLTK (Lemmatization) by Lathika Kotian - 98  
  
Socks : sock  
corpora : corpus  
better : good
```


PRACTICAL - 5

Aim: Write a program to implement the N-gram model

NLTK Code: TriGram Model

```
#5.) N-GRAM Model -Trigram
print("PRACTICAL-5 NLTK (N-GRAM Model) by Lathika Kotian - 98\n")
import nltk
nltk.download('punkt')
from nltk.util import ngrams
from nltk.tokenize import word_tokenize
data = "The movie was exciting and thrilling."
tokens = nltk.word_tokenize(data)
Ngram = ngrams(tokens, 3)
print("TriGram Model : ")
for gram in Ngram:
    print("\t", gram)
```

Output:

```
PRACTICAL-5 NLTK (N-GRAM Model) by Lathika Kotian - 98

TriGram Model :
    ('The', 'movie', 'was')
    ('movie', 'was', 'exciting')
    ('was', 'exciting', 'and')
    ('exciting', 'and', 'thrilling')
```

NLTK Code: BiGram Model

```
#5.) N-GRAM Model-Bigram
print("PRACTICAL-5 NLTK (N-GRAM Model for n=2) by Lathika Kotian - 98\n")
import nltk
```

```
nltk.download('punkt')
from nltk.util import ngrams
from nltk.tokenize import word_tokenize
data = "The movie was exciting and thrilling."
tokens = nltk.word_tokenize(data)
Ngram = ngrams(tokens, 2)
print("BiGram Model : ")
for gram in Ngram:
    print("\t", gram)
```

Output:

```
PRACTICAL-5 NLTK (N-GRAM Model for n=2) by Lathika Kotian - 98

BiGram Model :
    ('The', 'movie')
    ('movie', 'was')
    ('was', 'exciting')
    ('exciting', 'and')
    ('and', 'thrilling')
```

Spacy Code:

```
#5.) N-GRAM Model using SpaCy
print("PRACTICAL-5 spaCy (N-GRAM Model) by Lathika Kotian - 98\n")
import spacy
from spacy.util import minibatch

nlp = spacy.load("en_core_web_sm")

data = "The movie was exciting and thrilling"
doc = nlp(data)

tokens = [token.text for token in doc] # Convert tokens to list of strings
n = 3

ngrams = zip(*[tokens[i:] for i in range(n)])
print("TriGram Model : ")
for gram in ngrams:
    print("\t", gram)
```

Output:

```
PRACTICAL-5 spaCy (N-GRAM Model) by Lathika Kotian - 98

TriGram Model :
    ('The', 'movie', 'was')
    ('movie', 'was', 'exciting')
    ('was', 'exciting', 'and')
    ('exciting', 'and', 'thrilling')
```

PRACTICAL - 6

Aim: Write a program to implement POS tagging

NLTK Code:

```
print("PRACTICAL-6 NLTK (POS Tagging) by Lathika Kotian - 98\n")
import nltk
from nltk.tokenize import word_tokenize
from nltk import pos_tag
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
text = "The quick brown fox jumps over the lazy dog."
words = word_tokenize(text)
print("Tokens/Words :-", words)

pos_tags = pos_tag(words)
print("Word with POS :-", pos_tags)
for word, tag in pos_tags:
    print(f"{word} : {tag}")

# Create and display a POS-tagged tree structure
tree_obj = nltk.Tree('Sentence', [(word, tag) for word, tag in pos_tags])
tree_obj.pretty_print()
```

Output:

```
PRACTICAL-6 NLTK (POS Tagging) by Lathika Kotian - 98

Tokens/Words :- ['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog', '.']
Word with POS :- [('The', 'DT'), ('quick', 'JJ'), ('brown', 'NN'), ('fox', 'NN'), ('jumps', 'VBZ'), ('ov
The : DT
quick : JJ
brown : NN
fox : NN
jumps : VBZ
over : IN
the : DT
lazy : JJ
dog : NN
. : .

                        Sentence
                        |
_____The/DT quick/JJ brown/NN fox/NN jumps/VBZ over/IN the/DT lazy/JJ dog/NN ./._____
```

Spacy Code:

```
print("PRACTICAL-6 SpaCy (POS Tagging) by Lathika Kotian - 98\n")
import spacy
nlp = spacy.load('en_core_web_sm')

text = "The quick brown fox jumps over the lazy dog."

doc = nlp(text)
print("Tokens/Words :-", [token.text for token in doc])

print("\nWord with POS :-")
for token in doc:
    print(f"{token.text} : {token.pos_}")

# Visualize the POS tagging tree (syntax tree)
from spacy import displacy

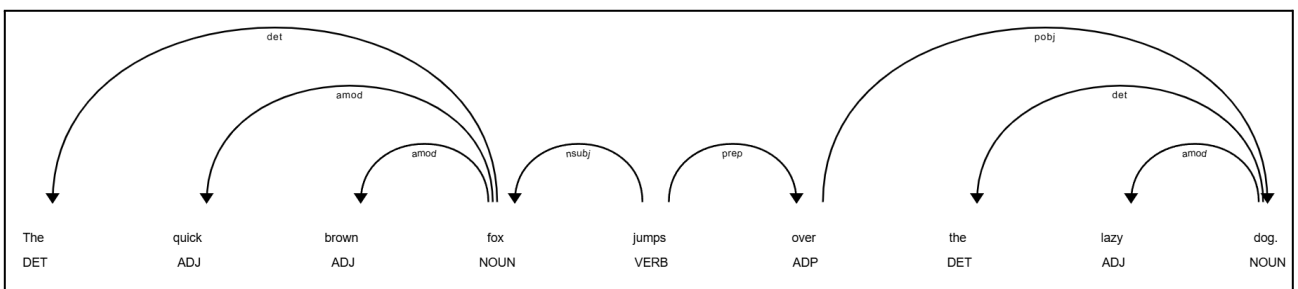
displacy.serve(doc, style='dep')
```

Output:

```
PRACTICAL-6 SpaCy (POS Tagging) by Lathika Kotian - 98

Tokens/Words :- ['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog', '.']

Word with POS :-
The : DET
quick : ADJ
brown : ADJ
fox : NOUN
jumps : VERB
over : ADP
the : DET
lazy : ADJ
dog : NOUN
. : PUNCT
```



PRACTICAL - 7

Aim: Write a program to build a custom NER system

NLTK Code:

```
#Name_entity_recognition
print("PRACTICAL-7 NLTK (NER) by Lathika Kotian - 98\n")
import nltk
from nltk import word_tokenize, pos_tag, ne_chunk
nltk.download('punkt')
nltk.download('maxent_ne_chunker')
nltk.download('words')
nltk.download('averaged_perceptron_tagger')
text = "Lathika Kotian was born in Mumbai, India on November 09, 2003. "
# Tokenizing and tagging
tokens = word_tokenize(text)
tagged_tokens = pos_tag(tokens)
named_entities = ne_chunk(tagged_tokens)
#print(named_entities)

from nltk.tree import Tree

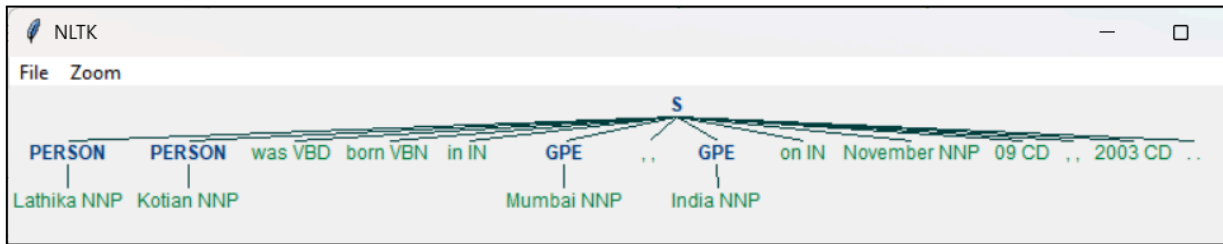
print("Named Entities:\n")
for subtree in named_entities:
    if isinstance(subtree, Tree):
        entity_name = " ".join(token for token, pos in subtree.leaves())
        entity_type = subtree.label()
        print(f"{entity_name} -> {entity_type}")

named_entities.draw()
```

Output:

```
PRACTICAL-7 NLTK (NER) by Lathika Kotian - 98

Named Entities:
    Lathika -> PERSON
    Kotian -> PERSON
    Mumbai -> GPE
    India -> GPE
```



SpaCy Code:

```
#Name_entity_recognition
print("PRACTICAL-7 SpaCy (NER) by Lathika Kotian - 98\n")
import spacy
from spacy import displacy

nlp = spacy.load("en_core_web_sm")

text = "Lathika Kotian was born in Mumbai, India on November 09, 2003."
doc = nlp(text)

print("Named Entities:")
for ent in doc.ents:
    print(f"\t{ent.text} -> {ent.label_}")
print()
# Visualize named entities in Colab
displacy.render(doc, style="ent", jupyter=True)
```

Output:

```
PRACTICAL-7 SpaCy (NER) by Lathika Kotian - 98

Named Entities:
    Lathika Kotian -> PERSON
    Mumbai -> GPE
    India -> GPE
    November 09, 2003 -> DATE
```

Lathika Kotian **PERSON** was born in **Mumbai GPE** , **India GPE** on **November 09, 2003 DATE**

PRACTICAL - 8A

Aim: Write a program to create a bag of words(bow) text representation.

NLTK Code:

```
print("PRACTICAL-8A NLTK (BOW) by Lathika Kotian - 98\n")
# Bag of words
import nltk
import numpy as np
from collections import Counter
nltk.download('punkt')

texts = [
    "I like to play badminton.",
    "Badminton is a great sport.",
]

# Tokenize the texts
tokenized_texts = [nltk.word_tokenize(text.lower()) for text in texts]

# Create a vocabulary (set of all unique words)
vocabulary = set(word for text in tokenized_texts for word in text)
print("Vocabulary:", vocabulary)

# Bag of Words (BoW) representation
def get_bow_representation(tokens, vocabulary):
    return [tokens.count(word) for word in vocabulary]

bow_vectors = [get_bow_representation(text, vocabulary) for text in
tokenized_texts]

# Print BoW vectors
print("\nBoW vectors:")
for i, vector in enumerate(bow_vectors):
    print(f"Sentence {i+1}: {vector}")
```


Output:

```
PRACTICAL-8A NLTK (BOW) by Lathika Kotian - 98

Vocabulary: {'play', 'i', 'to', 'badminton', 'sport', 'great', 'is', 'like', 'a'}

BoW vectors:
    Sentence 1: [1, 1, 1, 1, 0, 0, 0, 1, 0]
    Sentence 2: [0, 0, 0, 1, 1, 1, 1, 0, 1]
```

SpaCy Code:

```
# Bag of words
print("PRACTICAL-8A SpaCy(BOW) by Lathika Kotian - 98\n")

import spacy
import numpy as np
nlp = spacy.load("en_core_web_sm")

texts = [
    "I like to play badminton.",
    "Badminton is a great sport.",
]

# Process the texts using spaCy
processed_texts = [nlp(text.lower()) for text in texts]

# Create vocabulary: unique words (without stopwords or punctuation)
vocabulary = sorted(set(token.text for doc in processed_texts for token in doc if
token.is_alpha and not token.is_stop))
print("Vocabulary:", vocabulary)

# Create Bag of Words (BoW) representation
def get_bow_representation(doc, vocabulary):
    return [doc.text.count(word) for word in vocabulary]

# Create BoW vectors for each sentence
bow_vectors = [get_bow_representation(doc, vocabulary) for doc in
processed_texts]
```

```
# Print BoW vectors
print("\nBoW vectors:")
for i, vector in enumerate(bow_vectors):
    print(f"Sentence {i+1}: {vector}")
```

Output:

```
PRACTICAL-8A SpaCy(BOW) by Lathika Kotian - 98

Vocabulary: ['badminton', 'great', 'like', 'play', 'sport']

BoW vectors:
    Sentence 1: [1, 0, 1, 1, 0]
    Sentence 2: [1, 1, 0, 0, 1]
```

PRACTICAL - 8B

Aim: Write a program to create TF_IDF Text Representations.

NLTK Code:

```
print("PRACTICAL-8B NLTK(TF-IDF) by Lathika Kotian - 98\n")
#PROGRAM FOR TF_IDF
import nltk
import numpy as np
import pandas as pd
from math import log

# Download tokenizer quietly
nltk.download('punkt', quiet=True)

# Input texts
texts = [
    "I like to play badminton.",
    "Badminton is a great sport.",
]

# Tokenize and lowercase
tokenized_texts = [nltk.word_tokenize(text.lower()) for text in texts]

# Create sorted vocabulary for consistent ordering
vocabulary = sorted(set(word for text in tokenized_texts for word in text))
print("Vocabulary:", vocabulary)

# TF function
def get_tf(tokens, vocabulary):
    return [tokens.count(word) for word in vocabulary]

# IDF function
def get_idf(vocabulary, docs):
    num_docs = len(docs)
    idf_vector = []
```

```
for word in vocabulary:
    num_docs_with_word = sum(1 for doc in docs if word in doc)
    idf_value = log(num_docs / (1 + num_docs_with_word)) + 1 # Smoothed
    idf_vector.append(idf_value)
return idf_vector

# TF-IDF function
def get_tfidf(tokens, vocabulary, idf_vector):
    tf_vector = get_tf(tokens, vocabulary)
    return [tf * idf for tf, idf in zip(tf_vector, idf_vector)]

# Compute IDF
idf_vector = get_idf(vocabulary, tokenized_texts)

# Show IDF nicely
idf_df = pd.DataFrame({
    'Word': vocabulary,
    'IDF': np.round(idf_vector, 4)
})
print("\nIDF Values:")
print(idf_df)

# Compute TF-IDF for each doc
tfidf_matrix = []
for tokens in tokenized_texts:
    tfidf = get_tfidf(tokens, vocabulary, idf_vector)
    tfidf_matrix.append(tfidf)

# Convert to DataFrame for display
tfidf_df = pd.DataFrame(np.round(tfidf_matrix, 4), columns=vocabulary,
index=[f"Doc{i+1}" for i in range(len(texts))])
print("\nTF-IDF Matrix:")
print(tfidf_df)
```

Output:

PRACTICAL-8B NLTK(TF-IDF) by Lathika Kotian - 98

Vocabulary: ['a', 'badminton', 'great', 'i', 'is', 'like', 'play', 'sport', 'to']

IDF Values:

	Word	IDF
0	a	1.0000
1	badminton	0.5945
2	great	1.0000
3	i	1.0000
4	is	1.0000
5	like	1.0000
6	play	1.0000
7	sport	1.0000
8	to	1.0000

TF-IDF Matrix:

	a	badminton	great	i	is	like	play	sport	to
Doc1	0.0	0.5945	0.0	1.0	0.0	1.0	1.0	0.0	1.0
Doc2	1.0	0.5945	1.0	0.0	1.0	0.0	0.0	1.0	0.0

SpaCy Code:

```
print("PRACTICAL-8B spaCy (TF-IDF) by Lathika Kotian - 98\n")

import spacy
import numpy as np
import pandas as pd
from math import log
nlp = spacy.load("en_core_web_sm")

texts = [
    "I like to play badminton.",
    "Badminton is a great sport.",
]

# Tokenize and lowercase using spaCy
tokenized_texts = [[token.text.lower() for token in nlp(text) if not token.is_punct
and not token.is_space] for text in texts]
```

```
# Create sorted vocabulary
vocabulary = sorted(set(word for text in tokenized_texts for word in text))
print("Vocabulary:", vocabulary)

# TF function
def get_tf(tokens, vocabulary):
    return [tokens.count(word) for word in vocabulary]

# IDF function
def get_idf(vocabulary, docs):
    num_docs = len(docs)
    idf_vector = []
    for word in vocabulary:
        num_docs_with_word = sum(1 for doc in docs if word in doc)
        idf_value = log(num_docs / (1 + num_docs_with_word)) + 1 # Smoothed
        idf_vector.append(idf_value)
    return idf_vector

# TF-IDF function
def get_tfidf(tokens, vocabulary, idf_vector):
    tf_vector = get_tf(tokens, vocabulary)
    return [tf * idf for tf, idf in zip(tf_vector, idf_vector)]

# Compute IDF
idf_vector = get_idf(vocabulary, tokenized_texts)

idf_df = pd.DataFrame({
    'Word': vocabulary,
    'IDF': np.round(idf_vector, 4)
})
print("\nIDF Values:")
print(idf_df)

# Compute TF-IDF for each doc
tfidf_matrix = []
```

```
for tokens in tokenized_texts:
    tfidf = get_tfidf(tokens, vocabulary, idf_vector)
    tfidf_matrix.append(tfidf)

# Convert to DataFrame for display
tfidf_df = pd.DataFrame(np.round(tfidf_matrix, 4), columns=vocabulary,
index=[f"Doc{i+1}" for i in range(len(texts))])
print("\nTF-IDF Matrix:")
print(tfidf_df)
```

Output:

```
PRACTICAL-8B spaCy (TF-IDF) by Lathika Kotian - 98

Vocabulary: ['a', 'badminton', 'great', 'i', 'is', 'like', 'play', 'sport', 'to']

IDF Values:
      Word    IDF
0      a  1.0000
1 badminton 0.5945
2    great  1.0000
3       i  1.0000
4      is  1.0000
5     like  1.0000
6     play  1.0000
7    sport  1.0000
8      to  1.0000

TF-IDF Matrix:
      a badminton great  i  is  like  play  sport  to
Doc1 0.0    0.5945  0.0  1.0  0.0  1.0  1.0  0.0  1.0
Doc2 1.0    0.5945  1.0  0.0  1.0  0.0  0.0  1.0  0.0
```

PRACTICAL - 8C

Aim: Write a program to compare two vectors of bow using cosine similarity.

NLTK Code:

```
#PROGRAM TO COMPARE TWO VECTORS OF BOW USING COSINE SIMILARITY
print("PRACTICAL-8c NLTK(COSINE Similarity) by Lathika Kotian - 98\n")

import nltk
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
nltk.download('punkt')
texts = [
    "I like to play badminton",
    "Badminton is a great sport"
]

# Tokenize the texts
tokenized_texts = [nltk.word_tokenize(text.lower()) for text in texts]

# Create a vocabulary (set of all unique words)
vocabulary = set(word for text in tokenized_texts for word in text)
print("Vocabulary: ",vocabulary)
# Bag of Words (BoW) representation
def get_bow_representation(tokens, vocabulary):
    return [tokens.count(word) for word in vocabulary]
bow_vectors = [get_bow_representation(text, vocabulary) for text in
tokenized_texts]

# Print BoW vectors
print("\nBag of Words (BoW) Vectors:")
for i, bow_vector in enumerate(bow_vectors, start=1):
    print(f"Text {i} BoW vector: {np.array(bow_vector)}")

bow_similarity = cosine_similarity([bow_vectors[0]], [bow_vectors[1]])[0][0]
```



```
print(f"\nCosine Similarity between Text 1 and Text 2 is {bow_similarity*100}%")
```

Output:

```
PRACTICAL-8c NLTK(COSINE Similarity) by Lathika Kotian - 98

Vocabulary:  {'play', 'i', 'to', 'badminton', 'sport', 'great', 'is', 'like', 'a'}

Bag of Words (BoW) Vectors:
Text 1 BoW vector: [1 1 1 1 0 0 0 1 0]
Text 2 BoW vector: [0 0 0 1 1 1 1 0 1]

Cosine Similarity between Text 1 and Text 2 is 20.0%
```

Spacy Code:

```
import spacy
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
nlp = spacy.load('en_core_web_sm')

print("PRACTICAL-8c Cosine Similarity using SpaCy by Lathika Kotian - 98\n")

texts = [
    "I like to play badminton",
    "Badminton is a great sport"
]

# Process the texts with SpaCy
docs = [nlp(text) for text in texts]

# Create a vocabulary (set of all unique words)
vocabulary = set(token.text.lower() for doc in docs for token in doc if not
token.is_punct)
print("Vocabulary: ", vocabulary)

# Bag of Words (BoW) representation
```

```
def get_bow_representation(doc, vocabulary):  
    return [doc.text.lower().split().count(word) for word in vocabulary]  
  
bow_vectors = [get_bow_representation(doc, vocabulary) for doc in docs]  
  
# Print BoW vectors  
print("\nBag of Words (BoW) Vectors:")  
for i, bow_vector in enumerate(bow_vectors, start=1):  
    print(f"Text {i} BoW vector: {np.array(bow_vector)}")  
  
bow_similarity = cosine_similarity([bow_vectors[0]], [bow_vectors[1]])[0][0]  
print(f"\nCosine Similarity between Text 1 and Text 2 is {bow_similarity*100}%")
```

Output:

```
PRACTICAL-8c Cosine Similarity using SpaCy by Lathika Kotian - 98  
  
Vocabulary: {'play', 'i', 'to', 'badminton', 'sport', 'great', 'is', 'like', 'a'}  
  
Bag of Words (BoW) Vectors:  
Text 1 BoW vector: [1 1 1 1 0 0 0 1 0]  
Text 2 BoW vector: [0 0 0 1 1 1 1 0 1]  
  
Cosine Similarity between Text 1 and Text 2 is 20.0%
```

PRACTICAL - 8D

Aim: Write a program to compare a bow vector with a tf-idf vector using cosine similarity.

NLTK Code:

```
print("PRACTICAL-8D (BOW + TF-IDF) Cosine Similarity using NLTK by Lathika Kotian  
- 98\n")  
  
#PROGRAM TO compare a BoW vector with a TF-IDF vector using cosine similarity  
import nltk  
import numpy as np  
from collections import Counter  
from math import log  
from sklearn.metrics.pairwise import cosine_similarity  
  
# Ensure you have the necessary NLTK resources  
nltk.download('punkt')  
texts = [  
    "I like to play badminton",  
    "Badminton is a great sport"  
]  
  
# Tokenize the texts  
tokenized_texts = [nltk.word_tokenize(text.lower()) for text in texts]  
  
# Create a vocabulary (set of all unique words)  
vocabulary = set(word for text in tokenized_texts for word in text)  
print("Vocabulary:", vocabulary)  
  
# Bag of Words (BoW) representation  
def get_bow_representation(tokens, vocabulary):  
    return [tokens.count(word) for word in vocabulary]  
  
bow_vectors = [get_bow_representation(text, vocabulary) for text in  
tokenized_texts]  
  
# Function to compute Term Frequency (TF)  
def get_tf(tokens, vocabulary):
```

```
return [tokens.count(word) for word in vocabulary]

def get_idf(vocabulary, docs):
    num_docs = len(docs)
    idf_vector = []
    for word in vocabulary:
        # Count the number of documents containing the word
        num_docs_with_word = sum(1 for doc in docs if word in doc)
        # Calculate IDF as log(num_docs / (1 + num_docs_with_word)) to avoid
division by zero
        idf_value = log(num_docs / (1 + num_docs_with_word)) + 1
        idf_vector.append(idf_value)
    return idf_vector

# Function to compute TF-IDF
def get_tfidf(tokens, vocabulary, idf_vector):
    tf_vector = get_tf(tokens, vocabulary)
    tfidf_vector = [tf * idf for tf, idf in zip(tf_vector, idf_vector)]
    return tfidf_vector

# Calculate IDF for the entire corpus
idf_vector = get_idf(vocabulary, tokenized_texts)

# Compute TF-IDF for each document
tfidf_vectors = [get_tfidf(text, vocabulary, idf_vector) for text in tokenized_texts]

# Compute cosine similarity between BoW and TF-IDF vectors for doc1
bow_similarity = cosine_similarity([bow_vectors[0]], [tfidf_vectors[0]])[0][0]
print(f"\nCosine Similarity between doc1 (BoW) and doc1
(TF-IDF):{bow_similarity*100:.2f}%")

# Compute cosine similarity between BoW and TF-IDF vectors for doc2
bow_similarity = cosine_similarity([bow_vectors[1]], [tfidf_vectors[1]])[0][0]
print(f"Cosine Similarity between doc2 (BoW) and doc2 (TF-IDF) :
```

```
{bow_similarity*100:.2f}%")
```

Output:

```
PRACTICAL-8D (BOW + TF-IDF) Cosine Similarity using NLTK by Lathika Kotian - 98  
Vocabulary: {'play', 'i', 'to', 'badminton', 'sport', 'great', 'is', 'like', 'a'}  
Cosine Similarity between doc1 (BoW) and doc1 (TF-IDF):98.48%  
Cosine Similarity between doc2 (BoW) and doc2 (TF-IDF) : 98.48%
```

SpaCy Code:

```
import spacy  
import numpy as np  
from collections import Counter  
from math import log  
from sklearn.metrics.pairwise import cosine_similarity  
nlp = spacy.load('en_core_web_sm')  
  
print("PRACTICAL-8D (BOW + TF-IDF) Cosine Similarity using SpaCy by Lathika  
Kotian - 98\n")  
  
texts = [  
    "I like to play badminton",  
    "Badminton is a great sport"  
]  
  
# Process the texts with SpaCy  
docs = [nlp(text) for text in texts]  
  
# Create a vocabulary (set of all unique words)  
vocabulary = set(token.text.lower() for doc in docs for token in doc if not  
token.is_punct)  
print("Vocabulary:", vocabulary)
```

Bag of Words (BoW) representation

```
def get_bow_representation(doc, vocabulary):  
    return [doc.text.lower().split().count(word) for word in vocabulary]  
  
bow_vectors = [get_bow_representation(doc, vocabulary) for doc in docs]
```

Function to compute Term Frequency (TF)

```
def get_tf(doc, vocabulary):  
    return [doc.text.lower().split().count(word) for word in vocabulary]
```

Function to compute Inverse Document Frequency (IDF)

```
def get_idf(vocabulary, docs):  
    num_docs = len(docs)  
    idf_vector = []  
    for word in vocabulary:  
        # Count the number of documents containing the word  
        num_docs_with_word = sum(1 for doc in docs if word in  
doc.text.lower().split())  
        # Calculate IDF as  $\log(\text{num\_docs} / (1 + \text{num\_docs\_with\_word}))$  to avoid  
division by zero  
        idf_value =  $\log(\text{num\_docs} / (1 + \text{num\_docs\_with\_word})) + 1$   
        idf_vector.append(idf_value)  
    return idf_vector
```

Function to compute TF-IDF

```
def get_tfidf(doc, vocabulary, idf_vector):  
    tf_vector = get_tf(doc, vocabulary)  
    tfidf_vector = [tf * idf for tf, idf in zip(tf_vector, idf_vector)]  
    return tfidf_vector
```

Calculate IDF for the entire corpus

```
idf_vector = get_idf(vocabulary, docs)
```

Compute TF-IDF for each document

```
tfidf_vectors = [get_tfidf(doc, vocabulary, idf_vector) for doc in docs]
```

```
# Compute cosine similarity between BoW and TF-IDF vectors for doc1
bow_similarity = cosine_similarity([bow_vectors[0]], [tfidf_vectors[0]])[0][0]
print(f"\nCosine Similarity between doc1 (BoW) and doc1 (TF-IDF):
{bow_similarity*100:.2f}%")

# Compute cosine similarity between BoW and TF-IDF vectors for doc2
bow_similarity = cosine_similarity([bow_vectors[1]], [tfidf_vectors[1]])[0][0]
print(f"Cosine Similarity between doc2 (BoW) and doc2 (TF-IDF):
{bow_similarity*100:.2f}%")
```

Output:

```
PRACTICAL-8D (BOW + TF-IDF) Cosine Similarity using SpaCy by Lathika Kotian - 98

Vocabulary: {'play', 'i', 'to', 'badminton', 'sport', 'great', 'is', 'like', 'a'}

Cosine Similarity between doc1 (BoW) and doc1 (TF-IDF): 98.48%
Cosine Similarity between doc2 (BoW) and doc2 (TF-IDF): 98.48%
```

PRACTICAL - 9

Aim: Write a Program for Training and using word embedding
WordToVec/GloVe

NLTK Code:

```
#PROGRAM FOR TRAINING AND USING WORD EMBEDDING WORDTOVEC
!pip install gensim
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
import nltk
nltk.download('punkt')

print("PRACTICAL-9 WordToVec using NLTK by Lathika Kotian - 98\n")
# Function to train Word2Vec model
def train_word_embeddings(sentences):
    # Tokenize sentences using NLTK word_tokenize and convert to lowercase
    tokenized_sentences = [word_tokenize(sentence.lower()) for sentence in
sentences]

    # Train Word2Vec model
    model = Word2Vec(sentences=tokenized_sentences, vector_size=100,
window=5, min_count=1, workers=4)

    return model

# Function to use trained Word2Vec model and find similar words
def use_word_embeddings(model, word, top_n=5):
    try:
        # Get the top N similar words to the input word
        similar_words = model.wv.most_similar(word, topn=top_n)
        print(f"Words most similar to '{word}':")
        for w, score in similar_words:
            print(f"{w}: {score:.4f}")
    except KeyError:
        print(f"'{word}' not in vocabulary")
```



```
# Example usage
sentences = [
    "The quick brown fox jumps over the lazy dog",
    "A fox is a cunning animal",
    "The dog barks at night",
    "Foxes and dogs are different species"
]

# Train Word2Vec model using the provided sentences
model = train_word_embeddings(sentences)

# Use the trained model to find words similar to "fox"
use_word_embeddings(model, "fox")
```

Output:

```
PRACTICAL-9 WordToVec using NLTK by Lathika Kotian - 98

Words most similar to 'fox':
at: 0.1607
dogs: 0.1593
barks: 0.1372
night: 0.1230
and: 0.0854
```

PRACTICAL - 10

Aim: Write a Program to Implement a text classifier using NaiveBayes with scikit-learn

Code:

```
print("PRACTICAL-10  text classifier NaiveBayes using scikit learn by Lathika Kotian  
- 98\n")  
import numpy as np  
from sklearn.model_selection import train_test_split  
from sklearn.feature_extraction.text import CountVectorizer  
from sklearn.naive_bayes import MultinomialNB  
from sklearn import metrics  
  
texts = [  
    "I love programming",  
    "Python is great",  
    "I hate bugs",  
    "Coding is fun",  
    "I love solving problems",  
    "I hate error messages",  
    "Programming is awesome",  
    "Debugging is boring",  
    "This is terrible",  
    "Very disappointed with the experience",  
    "Worst product ever",  
    "Such a bad service",  
    "Absolutely amazing experience",  
    "Fantastic service and support",  
    "Really enjoyed the performance",  
    "Not satisfied at all after this"  
]  
  
labels = [  
    'positive', 'positive', 'negative', 'positive', 'positive', 'negative', 'positive', 'negative',  
    'negative', 'negative', 'negative', 'negative',
```

```
'positive', 'positive', 'positive', 'negative'
]

# Vectorize the text using CountVectorizer
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(texts)
y = np.array(labels)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=42)

# Train the Naive Bayes classifier
classifier = MultinomialNB()
classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = classifier.predict(X_test)

# Evaluate the model
print(f"Accuracy: {metrics.accuracy_score(y_test, y_pred):.4f}")
print("\nConfusion Matrix:")
print(metrics.confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(metrics.classification_report(y_test, y_pred))

# Classify new text inputs
new_texts = [
    "I really enjoyed the movie",
    "This product is terrible and disappointing",
    "Absolutely loved it!",
    "Not satisfied at all"
]

new_texts_vectorized = vectorizer.transform(new_texts)
new_predictions = classifier.predict(new_texts_vectorized)
```

```
print("\nNew Text Classification Results:")
for text, label in zip(new_texts, new_predictions):
    print(f"Text: '{text}' => Predicted Label: '{label}'")
```

Output:

```
PRACTICAL-10    text classifier NaiveBayes using scikit learn by Lathika Kotian - 98

Accuracy: 0.7500

Confusion Matrix:
[[1 0]
 [1 2]]

Classification Report:

```

	precision	recall	f1-score	support
negative	0.50	1.00	0.67	1
positive	1.00	0.67	0.80	3
accuracy			0.75	4
macro avg	0.75	0.83	0.73	4
weighted avg	0.88	0.75	0.77	4

```

New Text Classification Results:
Text: 'I really enjoyed the movie' => Predicted Label: 'negative'
Text: 'This product is terrible and disappointing' => Predicted Label: 'negative'
Text: 'Absolutely loved it!' => Predicted Label: 'positive'
Text: 'Not satisfied at all' => Predicted Label: 'negative'
```

PRACTICAL - 11

Aim: Write a program to build a Sentiment analysis system

Code:

```
print("PRACTICAL-11 Sentiment Analysis by Lathika Kotian - 98\n")
import nltk
from nltk.sentiment import SentimentIntensityAnalyzer
import pandas as pd

nltk.download('vader_lexicon')
def analyze_sentiment(text):
    sia = SentimentIntensityAnalyzer()
    sentiment_scores = sia.polarity_scores(text)

    if sentiment_scores['compound'] >= 0.1:
        sentiment = "Positive"
    elif sentiment_scores['compound'] <= -0.1:
        sentiment = "Negative"
    else:
        sentiment = "Neutral"

    return sentiment, sentiment_scores

def analyze_sentiments(texts):
    results = []
    for text in texts:
        sentiment, scores = analyze_sentiment(text)
        results.append({
            'text': text,
            'sentiment': sentiment,
            'pos_score': scores['pos'],
            'neg_score': scores['neg'],
            'neu_score': scores['neu'],
            'compound_score': scores['compound']
        })
```

```

return pd.DataFrame(results)

texts = [
    "I absolutely love this product! It's amazing!",
    "This is the worst experience I've ever had.",
    "The movie was okay, nothing special.",
    "I'm feeling pretty neutral about the whole situation.",
    "The customer service was excellent and very helpful!"
]

results_df = analyze_sentiments(texts)
print(results_df)

```

Output:

PRACTICAL-11 Sentiment Analysis by Lathika Kotian - 98

	text	sentiment	pos_score	\
0	I absolutely love this product! It's amazing!	Positive	0.689	
1	This is the worst experience I've ever had.	Negative	0.000	
2	The movie was okay, nothing special.	Neutral	0.233	
3	I'm feeling pretty neutral about the whole sit...	Positive	0.439	
4	The customer service was excellent and very he...	Positive	0.541	

	neg_score	neu_score	compound_score
0	0.000	0.311	0.8713
1	0.369	0.631	-0.6249
2	0.277	0.490	-0.0920
3	0.000	0.561	0.5719
4	0.000	0.459	0.7955

PRACTICAL - 12

Aim: Write a Program to create a text summarization tool

Code:

```
# PRACTICAL-1: Text summarization by Lathika Kotian - 98

from transformers import pipeline

print("PRACTICAL-12 Text summarization by Lathika Kotian - 98\n")

def summarize_text(text, max_length=150, min_length=50):
    summarizer = pipeline("summarization", model="facebook/bart-large-cnn")
    summary = summarizer(text, max_length=max_length, min_length=min_length,
do_sample=False)
    return summary[0]['summary_text']

long_text = """
Natural Language Processing is a subfield of artificial intelligence (AI) that focuses
on enabling machines to understand, interpret, and generate human language. NLP
combines computational linguistics, computer science, and machine learning to
process and analyze vast amounts of natural language data. As communication
through text and speech is fundamental to human interaction, NLP plays a crucial
role in bridging the gap between human and machine communication. One of the
fundamental tasks in NLP is tokenization, where text is broken down into smaller
units such as words or sentences. This is often the first step in more complex
processes like part-of-speech tagging, which assigns grammatical categories (like
noun or verb) to each word. Other essential techniques include stemming and
lemmatization, which reduce words to their root or dictionary form to help
normalize variations. More advanced NLP tasks include named entity recognition
(NER), which identifies proper nouns like names of people, organizations, and
locations, and sentiment analysis, which determines the emotional tone behind a
body of text. Text classification, another key task, allows for organizing documents
or messages into categories such as spam vs. non-spam, or positive vs. negative
reviews. With the advent of deep learning, NLP has advanced dramatically.
Particularly transformer-based architectures like BERT (Bidirectional Encoder
```

Representations from Transformers), GPT (Generative Pre-trained Transformer), and RoBERTa. These models are capable of understanding context, ambiguity, and even generating human-like text with impressive fluency. NLP is widely used in both consumer and enterprise applications. Popular use cases include chatbots and virtual assistants like Siri, Alexa, and Google Assistant, which rely on NLP to interpret and respond to voice commands.

```
"""
```

```
# Run summarization
```

```
summary = summarize_text(long_text)
```

```
# Output summary
```

```
print("Original text length:", len(long_text))
```

```
print("Summary length:", len(summary))
```

```
print("\nSummary:")
```

```
print(summary)
```

Output:

```
PRACTICAL-12 Text summarization by Lathika Kotian - 98
```

```
Device set to use cpu
```

```
Original text length: 1866
```

```
Summary length: 368
```

```
Summary:
```

```
Natural Language Processing is a subfield of artificial intelligence (AI) that focuses on enabling
```