**Tokenization**

#using nltk

import nltk

from nltk.tokenize import word\_tokenize,sent\_tokenize

nltk.download("punkt")

nltk.download("punkt\_tab")

text="Good morning. welcome to nlp practicals "

sentences=sent\_tokenize(text)

print("Sentence Tokenization:",sentences)

words=word\_tokenize(text)

print("Word Tokenization:",words)

#using spacy

import spacy

nlp=spacy.load("en\_core\_web\_sm")

text="आपकेविचार आपके जीव न का िनर् ा ाण करतेहैं. यहााााँसंग्रह िकयेगए हाान विचारक ं के हजार ं प्रेरक कथन आपके जीव न र् े ं एक सकारात्मक बदलाव ला सकतेहैं."

doc=nlp(text)

tokens=[token.text for token in doc]

print("Spacy Tokenization:",tokens) #word token

tokens=[sent.text for sent in doc.sents]

print("Spacy sentence Tokenization:",tokens) #sentence token

#using regular expression

import re

text="Hello. How are you?"

tokens=re.findall(r'\w+',text)

print("Regular Expression Tokenization:",tokens)

sentences=re.split(r"(?<=[.!?])\s+",text)

print("regex sentence tokenization:",sentences)

**Stop word**

import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

nltk.download('stopwords')

nltk.download('punkt')

nltk.download('punkt\_tab')

text="This is a simple example of stopword removal in nlp."

words=word\_tokenize(text)

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

filtered\_word=[word for word in words if word.lower() not in stop\_words]

print("Original Text:",words)

print("Filtered Text:",' ',filtered\_word)

**Stemming**

#porter and lancaster stemmer

import nltk

from nltk.stem import PorterStemmer

from nltk.stem import LancasterStemmer

from nltk.tokenize import word\_tokenize

nltk.download('punkt\_tab')

text="Cats are very friendly and loving animals. Dogs are very loyal to humans."

words=word\_tokenize(text)

porter=PorterStemmer()

lancaster=LancasterStemmer()

porter\_stemmed=[porter.stem(word) for word in words]

lancaster\_stemmed=[lancaster.stem(word) for word in words]

print("Original words:",words)

print("Porter Stemmed:",porter\_stemmed)

print("Lancaster Stemmed:",lancaster\_stemmed)

#Snowball stemmer

import nltk

from nltk.stem import SnowballStemmer

from nltk.tokenize import word\_tokenize

nltk.download('punkt\_tab')

text="Cats are very friendly and loving animals. Dogs are very loyal to humans."

words=word\_tokenize(text)

snowball=SnowballStemmer("english")

snowball\_stemmed=[snowball.stem(word) for word in words]

print("Original words:",words)

print("Snowball Stemmed:",snowball\_stemmed)

**Lemmatization**

import nltk

from nltk.corpus import wordnet

from nltk import pos\_tag

from nltk.tokenize import word\_tokenize

from nltk.stem import WordNetLemmatizer

nltk.download('wordnet')

nltk.download('averaged\_perceptron\_tagger')

nltk.download('averaged\_perceptron\_tagger\_eng')

nltk.download('punkt\_tab')

lemmatizer = WordNetLemmatizer()

def get\_wordnet\_pos(word):

  """Map NLTK POS tags to WordNet POS tags."""

  tag = pos\_tag([word])[0][1][0].upper()

  tag\_dict = {"J": wordnet.ADJ, "N": wordnet.NOUN, "V": wordnet.VERB, "R":

              wordnet.ADV}

  return tag\_dict.get(tag, wordnet.NOUN)

sentence = "The striped bats are hanging on their feet for best"

words = word\_tokenize(sentence)

lemmatized\_words\_pos = [lemmatizer.lemmatize(word, get\_wordnet\_pos(word)) for word in words]

print("Lemmatized words with POS:", lemmatized\_words\_pos)

**N-Gram**

from nltk import word\_tokenize

from nltk.util import ngrams

import nltk

nltk.download('punkt\_tab')

text="A sample bigram model example. A sample trigram model."

tokens=word\_tokenize(text)

unigrams=list(ngrams(tokens,1))

tokens=word\_tokenize(text)

bigrams=list(ngrams(tokens,2))

tokens=word\_tokenize(text)

trigrams=list(ngrams(tokens,3))

print("Original text",text)

print("Unigram Text",unigrams)

print("Bigram Text",bigrams)

print("Trigram Text",trigrams)

**Pos tagging**

import spacy

nlp=spacy.load("en\_core\_web\_sm")

def pos\_tagging\_spacy(text):

  doc=nlp(text)

  return[(token.text,token.pos\_)for token in doc]

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

pos\_tags=pos\_tagging\_spacy(text)

print("POS Tags using spacy: ")

print(pos\_tags)

**NER**

import spacy

from spacy.training.example import Example

nlp=spacy.blank("en")

ner=nlp.add\_pipe("ner",last=True)

ner.add\_label("PERSON")

ner.add\_label("ORG")

TRAIN\_DATA=[

("Bill gates founded Microsoft.",{"entities":[(0,10,"PERSON"),(19,28,"ORG")]}),

("Elon Musk founded Tesla.",{"entities":[(0,10,"PERSON"),(19,24,"ORG")]}),

("Steve jobs created Apple",{"entities":[(0,10,"PERSON"),(19,24,"ORG")]})

]

optimizer= nlp.begin\_training()

for i in range (10):

  for text, annotations in TRAIN\_DATA:

    example=Example.from\_dict(nlp.make\_doc(text),annotations)

    nlp.update([example],sgd=optimizer)

for text,annotations in TRAIN\_DATA:

  doc=nlp.make\_doc(text)

  tags=spacy.training.offsets\_to\_biluo\_tags(doc,annotations.get("entities"))

  print(f"Text: {text}")

  print(f"Tags: {tags}")

**Text Representation(BOW)**

**1)**

#BoW

import nltk

import numpy as np

from collections import Counter

nltk.download('punkt\_tab')

texts=[

"The cat sat on the mat",

"The dog sat on the log"

]

tokenized\_texts=[nltk.word\_tokenize(text.lower()) for text in texts]

vocabulary=set(word for text in tokenized\_texts for word in text)

vocabulary\_size=len(vocabulary)

print(vocabulary)

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(np.array(bow\_vectors))

**2)**

#BoW

import nltk

import numpy as np

from collections import Counter

from math import log

nltk.download('punkt\_tab')

texts=[

"The cat sat on the mat",

"The dog sat on the log"

]

tokenized\_texts=[nltk.word\_tokenize(text.lower()) for text in texts]

vocabulary=sorted(list(set(word for text in tokenized\_texts for word in text)))

vocabulary\_size=len(vocabulary)

print("Vocabulary:", vocabulary)

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(np.array(bow\_vectors))

def get\_tf(tokens,vocabulary):

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

def get\_idf(vocabulary,docs):

  idf\_vector=[]

  for word in vocabulary:

    num\_docs\_with\_word=sum(1 for doc in docs if word in doc)

    idf\_value=log(len(docs)/(1+num\_docs\_with\_word))+1

    idf\_vector.append(idf\_value)

  return idf\_vector

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

idf\_vector = get\_idf(vocabulary, tokenized\_texts)

tfidf\_vectors=[get\_tfidf(text,vocabulary,idf\_vector) for text in tokenized\_texts]

print("\n TF-IDF vectors:")

print(np.array(tfidf\_vectors))

**3)**

import nltk

import numpy as np

from sklearn.metrics.pairwise import cosine\_similarity

nltk.download('punkt\_tab')

texts=[

"The cat sat on the mat.",

"The mat is on the table."

]

tokenized\_texts=[nltk.word\_tokenize(text.lower()) for text in texts]

vocabulary=set(word for text in tokenized\_texts for word in text)

print(vocabulary)

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(np.array(bow\_vectors))

bow\_similarity=cosine\_similarity([bow\_vectors[0]],

[bow\_vectors[1]])[0][0]

print("bow\_similarity: ")

print(bow\_similarity)

**4)**

import nltk

import numpy as np

from collections import Counter

from math import log

from sklearn.metrics.pairwise import cosine\_similarity

nltk.download('punkt\_tab')

texts=[

"The cat sat on the mat.",

"The mat is on the table."

]

tokenized\_texts=[nltk.word\_tokenize(text.lower()) for text in texts]

vocabulary=set(word for text in tokenized\_texts for word in text)

print(vocabulary)

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]

def get\_tf(tokens,vocabulary):

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

def get\_idf(vocabulary,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\_with\_word/(1+num\_docs\_with\_word))+1

    idf\_vector.append(idf\_value)

  return idf\_vector

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

idf\_vector = get\_idf(vocabulary, tokenized\_texts)

print("\n IDF vector")

print(idf\_vector)

tfidf\_vectors=[get\_tfidf(text,vocabulary,idf\_vector) for text in tokenized\_texts]

bow\_similarity=cosine\_similarity([bow\_vectors[0]],

[tfidf\_vectors[1]])[0][0]

print("Cosine similarity between doc1(Bow) and doc2(TF-IDF):")

print(bow\_similarity)

bow\_similarity=cosine\_similarity([bow\_vectors[1]],

[tfidf\_vectors[0]])[0][0]

print("Cosine similarity between doc1(Bow) and doc2(TF-IDF):")

print(bow\_similarity)

**Text classifier**

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.naive\_bayes import MultinomialNB

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report

def train\_test\_classifier(X,y):

  X\_train,X\_test,y\_train,y\_test =train\_test\_split(X,y,test\_size=0.2,random\_state=42)

  vectorizer=CountVectorizer()

  X\_train\_vectorized=vectorizer.fit\_transform(X\_train)

  X\_test\_vectorized=vectorizer.transform(X\_test)

  classifier=MultinomialNB()

  classifier.fit(X\_train\_vectorized,y\_train)

  y\_pred=classifier.predict(X\_test\_vectorized)

  print(classification\_report(y\_test,y\_pred))

  return vectorizer,classifier

def classify\_text(text,vectorizer,classifier):

  text\_vectorized=vectorizer.transform([text])

  prediction=classifier.predict(text\_vectorized)

  return prediction[0]

X=[

"My mother cooks a very delicious pizza.",

"My father is not going to his office.",

"I love the gaming laptop my brother bought.",

"This movie is amazing.",

"She hates playing tennis with her classmate.",

"Today is a beautiful day.",

"This movie is horrible."

]

y=["positive","negative","positive","positive","negative","positive","negative"]

new\_text="I love this food."

vectorizer,classifier=train\_test\_classifier(X,y)

prediction=classify\_text(new\_text,vectorizer,classifier)

print(f"Prediction for'{new\_text}':{prediction}")

**Sentiment Analysis**

import nltk

from nltk.sentiment import SentimentIntensityAnalyzer

import pandas as pd

# Ensure the vader\_lexicon is downloaded

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"

  # Corrected indentation: return statement is now outside the if/elif/else blocks

  return sentiment, sentiment\_scores

def analyze\_sentiments(texts):

  results = []

  for text in texts:

    # Call the corrected analyze\_sentiment function

    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']

    })

  # The return for analyze\_sentiments should be outside the loop

  return pd.DataFrame(results)

texts=[

"My mother cooks a very delicious pizza.",

"My father is not going to his office.",

"I love the gaming laptop my brother bought.",

"This movie is amazing.",

"She hates playing tennis with her classmate.",

"Today is a beautiful day.",

"This movie is horrible."

]

# Call the analyze\_sentiments function and print the result

results\_df = analyze\_sentiments(texts)

print(results\_df)

**Text Summarization**

!pip install transformers

!pip install torch

from transformers import pipeline

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 = """

Climate change is one of the most pressing issues facing our planet today. It refers to longterm shifts in temperatures and weather patterns, mainly caused by human activities,

especially the burning of fossil fuels. These activities release greenhouse gases into the

atmosphere, trapping heat and causing the Earth's average temperature to rise.

The consequences of climate change are far-reaching and include more frequent and severe

weather events, rising sea levels, and disruptions to ecosystems. To address this global

challenge, countries and organizations worldwide are working on strategies to reduce

greenhouse gas emissions and transition to cleaner energy sources.

Individual actions, such as reducing energy consumption and adopting sustainable practices,

also play a crucial role in mitigating the effects of climate change."""

summary = summarize\_text(long\_text)

print("Original text length:", len(long\_text))

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

print("\nSummary:")

print(summary)