NATURAL LANGUAGE PROCESSING - LAB

(COURSE CODE: 23UPCSC4E17)

A Laboratory record submitted to Periyar University, Salem.
In partial fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE
IN
DATA SCIENCE

BY

GOWTHAM S
REG NO: U23PG507DTS010

DEPARTMENT OF COMPUTER SCIENCE

PERIYAR UNIVERSITY

PERIYAR PALKALAI NAGAR,
SALEM – 636011

CERTIFICATE

This is to certify that the Programming Laboratory entitled **"NATURAL LANGUAGE PROCESSING - LAB"** (23UPCSC4E17) is a bonafide record work done by **Mr. GOWTHAM S** Register No. **U23PG507DTS010** as partial fulfillment of the requirements degree of **MASTER OF SCIENCE IN DATA SCIENCE** in the Department of Computer Science, Periyar University, Salem, during the year 2023 – 2025.

Faculty In-Charge Head of the Department

Submitted for the practical examination held on ……./……./……

Marks Obtained

Internal Examiner External Examiner

CERTIFICATION

Infosys®
Navigate your next

COURSE COMPLETION CERTIFICATE

The certificate is awarded to

GOWTHAM SARGUNAM

for successfully completing the course

Introduction to Natural Language Processing

on August 19, 2024

Infosys | Springboard

Congratulations! You make us proud!

Issued on: Monday, August 19, 2024
To verify, scan the QR code at https://verify.onwingspan.com

Thirumala Arohi
Executive Vice President and Global Head
Education, Training & Assessment (ETA)
Infosys Limited

INDEX

EX.NO:1	Fundamentals of NLP I Tokenization & Lemmatization
Date:03-07-2024	

AIM:

To do tokenize and lemmatize the words and paragraphs

THEORY:

Tokenization in NLP is the process of splitting text into smaller units, such as words or phrases, called tokens. This is essential for further text processing tasks.
Lemmatization reduces words to their base or dictionary form, called a "lemma," by considering the context and meaning of the word.

PROCEDURE:

- Pip install libraries
- Import libraries
- Tokenization
- Lemmatization

Pip install libraries

```
pip install nltk
```

```
Requirement already satisfied:
Requirement already satisfied:
```

Import libraries

```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Gowth\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
[3]:
```

```
True
```

```
[5]:
```

```
from nltk import word_tokenize, sent_tokenize
```

Tokenization:

There are several types of tokenization techniques, each useful for different tasks in Natural Language Processing (NLP). Here are the main types:

Word Tokenization
Sentence Tokenization

i. Word Tokenization

```
text = "The fifth of November"

# Word Tokenization
word_tokens = word_tokenize(text)
print("Word Tokens:", word_tokens)
```

```
Word Tokens: ['The', 'fifth', 'of', 'November']
```

ii. Sentence Tokenization

```
te = "don't live your past! live your life!"

# Sentence Tokenization
sentence_tokens = sent_tokenize(te)
print("Sentence Tokens:", sentence_tokens)
```

```
Sentence Tokens: ["don't live your past!", 'live your life!']
```

Lemmatization:

Lemmatization is the process of reducing a word to its base or root form, considering its meaning and context.

i. WordNet Lemmatizer:

```
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

print("rocks :", lemmatizer.lemmatize("rocks"))
print("corpora :", lemmatizer.lemmatize("corpora"))
```

```
rocks : rock
corpora : corpus
```

ii. SpaCy Lemmatizer:

```
!python -m spacy download en_core_web_sm
```

```
Collecting en-core-web-sm==3.8.0
  Downloading https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-3.8.0/en_core_web_sm-3.8.0-py3-none-any.whl (12.8 MB)
     ---------------------------------------- 0.0/12.8 MB ? eta -:--:--
     - -------------------------------------- 0.5/12.8 MB 3.4 MB/s eta 0:00:04
     -- ------------------------------------- 0.8/12.8 MB 2.8 MB/s eta 0:00:05
     ---- ----------------------------------- 1.6/12.8 MB 3.0 MB/s eta 0:00:04
     -------- ------------------------------- 2.6/12.8 MB 3.5 MB/s eta 0:00:03
     ------------ --------------------------- 4.2/12.8 MB 4.3 MB/s eta 0:00:02
     --------------- ------------------------ 5.2/12.8 MB 4.5 MB/s eta 0:00:02
     -------------------- ------------------- 6.8/12.8 MB 5.0 MB/s eta 0:00:02
     ------------------------- -------------- 8.4/12.8 MB 5.4 MB/s eta 0:00:01
     ---------------------------- ---------- 9.4/12.8 MB 5.6 MB/s eta 0:00:01
     ------------------------------- -------- 10.2/12.8 MB 5.1 MB/s eta 0:00:01
     ---------------------------------------- - 12.3/12.8 MB 5.5 MB/s eta 0:00:01
     ---------------------------------------- 12.8/12.8 MB 5.6 MB/s eta 0:00:00
Installing collected packages: en-core-web-sm
Successfully installed en-core-web-sm-3.8.0
[+] Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
```

[11]:

```
n=spacy.load('en_core_web_sm')
```

[12]:

```
doc = n("running")
for token in doc:
    lemma = token.lemma_
    print(lemma)
```

run

RESULT:

Thus the Tokenization & Lemmatization was successfully executed

Ex. no: 2	Fundamentals of NLP II Stemming & Sentence Segmentation
DATE:09-07-2024	

AIM:

To implement process of stemming and sentence segmentation

THEORY:

- **Stemming** is the process of reducing words to their root form by stripping suffixes (e.g., "running" to "run"). Common types include Porter, Snowball, Lancaster, and Regexp stemmers.
- **Sentence segmentation** splits text into individual sentences, often using libraries like NLTK's sent_tokenize().

Procedure:

- Stemming.
- Sentence Segmentation

Stemming:

i. Porter Stemmer

Porter Stemmer is one of the most commonly used stemming algorithms. It applies a series of rules to strip suffixes from words.

```
from nltk.stem import PorterStemmer
ps = PorterStemmer()
words = ["running", "runner", "ran", "easily", "fairly"]
for word in words:
    print(f"{word} > {ps.stem(word)}")
```

```
running > run
runner > runner
ran > ran
easily > easili
fairly > fairli
```

ii. Lancaster Stemmer

Lancaster Stemmer is more aggressive in reducing words than Porter.

```
from nltk.stem import LancasterStemmer
ls = LancasterStemmer()
words = ["running", "runner", "ran", "easily", "fairly"]
for word in words:
    print(f"{word} > {ls.stem(word)}")
```

```
running > run
runner > run
ran > ran
easily > easy
fairly > fair
```

iii. Snowball Stemmer

It's also known as the Porter2 Stemmer is an improved version of the Porter Stemmer. It's more flexible and supports multiple languages. The Snowball stemmer is a popular choice for stemming due to its balance between speed and accuracy.

Note: Here I'm Using **Spanish (Español)**

```
from nltk.stem import SnowballStemmer

spanish_stemmer = SnowballStemmer(language="spanish")
spanish_words = ["corriendo", "corrió", "corre", "fácilmente", "felizmente"]
for word in spanish_words:
    print(f"{word} > {spanish_stemmer.stem(word)}")
```

```
corriendo > corr
corrió > corr
corre > corr
fácilmente > facil
felizmente > feliz
```

iv. Reg exp Stemmer:

The Regexp Stemmer is a rule-based stemmer that removes suffixes from words using regular expressions. Unlike other stemmers like Porter or Snowball, Regexp Stemmer does not follow a set of linguistic rules but instead relies on simple pattern matching.

```python
from nltk.stem import RegexpStemmer

# Define a regexp pattern to match common suffixes
regexp_stemmer = RegexpStemmer('ing$|ed$|ly$', min=4)

# List of words
words = ["running", "jogged", "happily", "easily", "fairly","eating"]

# Stemming words using RegexpStemmer
for word in words:
    print(f"{word} > {regexp_stemmer.stem(word)}")
```

```
running > runn
jogged > jogg
happily > happi
easily > easi
fairly > fair
eating > eat
```

Sentence Segmentation:

Sentence segmentation, also called sentence tokenization, splits a text into sentences. This can be done using various libraries like nltk.

```python
# Sample text
text = "Hello! How are you today? I'm doing well. Let's meet at 5 PM."
sentences = sent_tokenize(text)
for sentence in sentences:
    print(sentence)
```

```
Hello!
How are you today?
I'm doing well.
Let's meet at 5 PM.
```

RESULT:

Thus the Different types of Stemming and Sentence segmentation is successfully executed.

Ex.no: 3	NLP Using Scikit Library
DATE:16-07-2024	

AIM:

To create a program for NLP using Scikit library.

PROCEDURE:

1. Importing Libraries.
2. Loading Training and Testing Texts.
3. Preprocessing.
4. Model Building.
5. Classifying Report.

PROBLEM:

1. **Import libraries:**

```
import nltk
import re
```

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

```
from nltk.stem import PorterStemmer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
```

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

2. Loading Dataset:

```
import pandas as pd
train =pd.read_csv(r'D:\IMDB Dataset.csv')
train.head()
```

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

3. Preprocessing:

```
train.isnull().sum()
```

```
review       0
sentiment    0
dtype: int64
```

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   review     50000 non-null  object
 1   sentiment  50000 non-null  object
dtypes: object(2)
memory usage: 781.4+ KB
```

```
def preprocessing(text):
    text=text.lower()
    text=re.sub(r'[^a-zA-Z\s]','',text)
    words=word_tokenize(text)
    stop_words=set(stopwords.words('english'))
    words=[word for word in words if word not in stop_words]
    stemmer=PorterStemmer()
    words=[stemmer.stem(word) for word in words]
    return' '.join(words)
train['cleaned_review']=train['review'].apply(preprocessing)
train.head()
```

	review	sentiment	cleaned_review
0	One of the other reviewers has mentioned that ...	positive	one review mention watch oz episod youll hook ...
1	A wonderful little production. The...	positive	wonder littl product br br film techniqu unass...
2	I thought this was a wonderful way to spend ti...	positive	thought wonder way spend time hot summer weeke...
3	Basically there's a family where a little boy ...	negative	basic there famili littl boy jake think there ...
4	Petter Mattei's "Love in the Time of Money" is...	positive	petter mattei love time money visual stun film...

4. Model Building:

```
vectorizer=CountVectorizer()
X=vectorizer.fit_transform(train['cleaned_review'])

Y=train['sentiment']
```

```
x_train,x_valid,y_train,y_valid=train_test_split(X,Y,test_size=0.25,random_state=43)
model=RandomForestClassifier()
model.fit(x_train,y_train)
```

```
▾ RandomForestClassifier

RandomForestClassifier()
```

```
y_pred=model.predict(x_valid)
```

5. Accuracy Score:

```
accuracy=accuracy_score(y_valid,y_pred)
print(f"validation Accuracy:{accuracy*100:.2f}%")
```

```
validation Accuracy:85.11%
```

SVM accuracy score:

```
model_svm=SVC()
model_svm.fit(x_train,y_train)
```

```
▾ SVC

SVC()
```

```
y_pred=model_svm.predict(x_valid)
```

```
accuracy=accuracy_score(y_valid,y_pred)
print(f"validation Accuracy:{accuracy*100:.2f}%")
```

```
validation Accuracy:85.11%
```

RESULT:

Thus, the Dataset was completely analysed successfully using Scikit Library

Ex.no: 4	NLP using Spacy library
DATE:23-07-2024	

AIM:

To use spacy library for the task in nlp

THEORY:

Spacy is an open-source software library for advanced natural language processing, written in the programming languages Python and Cython.

PROCEDURE:

1. Sentence Detection.
2. Stop Words.
3. Word Frequency.
4. Part-of-speech Tagging.
5. Dependency Parsing

PROBLEM:

1. Importing Libraries and Data:

```
import spacy
```

```
[11]:
```

```
n=spacy.load('en_core_web_sm')
```

2. Sentence Detection:

```
: doc=nlp(u"you may need to restart. the kernel to use. Updated packages.")
for sent in doc.sents:
    print(sent)
```

```
you may need to restart.
the kernel to use.
Updated packages.
```

```
for text in doc:
    print(text)
```

```
you
may
need
to
restart
.
the
kernel
to
use
.
Updated
packages
.
```

3. Stop Words:

```
spacy_stopwords=spacy.lang.en.stop_words.STOP_WORDS
len(spacy_stopwords)
```

```
326
```

```
for stop_word in list(spacy_stopwords):
    print(stop_word)
```

```
on
eight
‘re
really
become
everywhere
they
does
under
just
whenever
‘s
thereby
beforehand
're
n’t
one
```

```
doc=nlp(texts)
fil_words=[token.text for token in doc if not token.is_stop]

print(fil_words)
```

```
['Sentence', 'detection', 'process', 'locating', 'sentences', 'start', 'end', 'given', 'text', '.', 'allows', 'divide', 'text',
'linguistically', 'meaningful', 'units', '.', 'use', 'units', 'processing', 'text', 'perform', 'tasks', '-', '-', 'speech',
'(', 'POS', ')', 'tagging', 'named', '-', 'entity', 'recognition', ',', 'come', 'later', 'tutorial', '.', 'spaCy', ',', '.sent
s', 'property', 'extract', 'sentences', 'Doc', 'object', '.', 'extract', 'total', 'number', 'sentences', 'sentences', 'given',
'input']
```

```
cln_sent=' '.join(fil_words)
print(cln_sent)
```

```
Sentence detection process locating sentences start end given text . allows divide text linguistically meaningful units . use u
nits processing text perform tasks - - speech ( POS ) tagging named - entity recognition , come later tutorial . spaCy , .sents
property extract sentences Doc object . extract total number sentences sentences given input
```

4. Part-of-speech Tagging:

```
doc=nlp(texts)
for token in doc:
    print(token.pos_,token.tag_,spacy.explain(token.pos_))
```

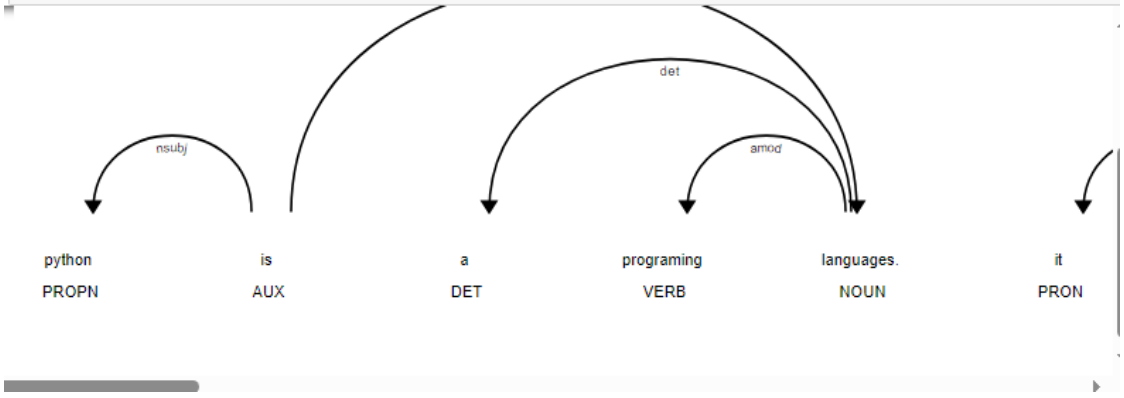
```
PROPN NNP proper noun
AUX VBZ auxiliary
DET DT determiner
VERB VBG verb
NOUN NNS noun
PUNCT . punctuation
PRON PRP pronoun
AUX VBZ auxiliary
ADV RB adverb
PART TO particle
VERB VB verb
DET DT determiner
NOUN NN noun
CCONJ CC coordinating conjunction
ADV RB adverb
PART TO particle
VERB VB verb
PUNCT . punctuation
PROPN NNP proper noun
AUX VBP auxiliary
VERB VBN verb
ADP IN adposition
NOUN NN noun
PUNCT , punctuation
NOUN NN noun
```

5. Dependency Parsing:

```
from spacy import displacy
doc=nlp(texts)
displacy.render(nlp(doc))
```

python PROPN is AUX a DET programing VERB languages. NOUN it PRON is AUX easily ADV to PART understand VERB the DET program NOUN and CCONJ easily ADV to PART download. VERB python PROPN are AUX used VERB in ADP idle, NOUN cmd NOUN and CCONJ also ADV etc. X nsubj det amod attr nsubj advmod aux xcomp det dobj cc advmod aux conj nsubjpass auxpass prep pobj conj cc advmod conj

```
from spacy import displacy
doc=nlp(texts)
displacy.render(doc,style='dep',jupyter=True)
```



RESULT:

Thus, spacy used and the codes are completed successfully

Ex. no: 5	WORKING WITH TF-IDF
DATE:30-07-2024	

AIM:

To do program for working with TF-IDF

PROCEDURE:

1. Importing Libraries.
2. Loading Training and Testing Texts.
3. Preprocessing.
4. Visualization
5. TF-IDF.
6. Model building

PROGRAM:

1. Importing libraries:

```
import seaborn as sns
import matplotlib.pyplot as plt
import os
import re
```

```
import spacy
from nltk.corpus import stopwords
from spacy import displacy
import nltk
```

```
from wordcloud import WordCloud
from nltk import word_tokenize
from nltk.stem import PorterStemmer
from sklearn.model_selection import train_test_split
```

```
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
```

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

```
from sklearn.metrics import confusion_matrix,classification_report
```

2. Loading Data:

```
import pandas as pd
train =pd.read_csv(r'D:\movies1.csv')
train.head()
```

user_name	review_id	review_title	review_content
Dragon,EyeDunno,ale...	rw2284594,rw6606154,rw1221355,rw1822343,rw1288...	Some birds aren't meant to be caged.,An incred...	The Shawshank Redemption is written and direct...
irgereviews,gogoschka-1,Sleepin...	rw3038370,rw4756923,rw4059579,rw6568526,rw1897...	The Pinnacle Of Flawless Films!,An offer so go...	'The Godfather' is the pinnacle of flawless fi...
_Cheese,dseferaj,little...	rw5478826,rw1914442,rw6606026,rw1917099,rw5170...	The Dark Knight,The Batman of our dreams! So m...	Confidently directed, dark, brooding, and pack...

3. Preprocessing Data:

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 250 entries, 0 to 249
Data columns (total 22 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   rank            250 non-null    int64
 1   movie_id        250 non-null    object
 2   title           250 non-null    object
 3   year            250 non-null    int64
 4   link            250 non-null    object
 5   imbd_votes      250 non-null    object
 6   imbd_rating     250 non-null    float64
 7   certificate     249 non-null    object
 8   duration        250 non-null    object
 9   genre           250 non-null    object
 10  cast_id         250 non-null    object
 11  cast_name       250 non-null    object
 12  director_id     250 non-null    object
 13  director_name   250 non-null    object
 14  writer_id       250 non-null    object
 15  writer_name     250 non-null    object
 16  storyline       250 non-null    object
 17  user_id         250 non-null    object
 18  user_name       250 non-null    object
 19  review_id       250 non-null    object
 20  review_title    250 non-null    object
 21  review_content  250 non-null    object
dtypes: float64(1), int64(2), object(19)
memory usage: 43.1+ KB
```

Removing non-characters from text:

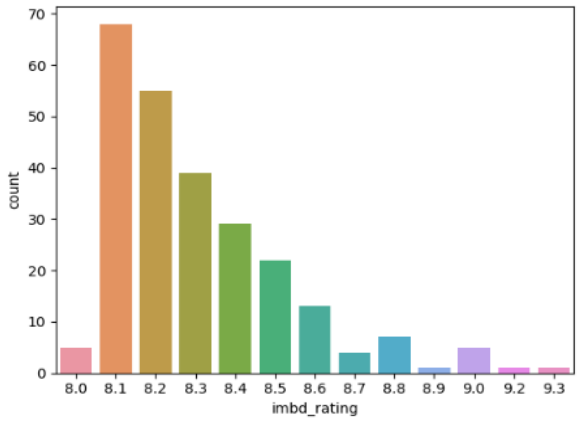
```
def preprocessing(text):
    text=text.lower()
    text=re.sub(r'[^a-zA-Z\s]','',text)
    text=re.sub(r'http\S+','',text)
    text=re.sub(r'\s+.com','',text)
    return text
```

```
train['review_content1']=train['review_content'].apply(preprocessing)
train.head()
```

genre	...	director_name	writer_id	writer_nar
Drama	...	Frank Darabont	nm0000175,nm0001104	Steph King,Fra Darabc
Crime,Drama	...	Francis Ford Coppola	nm0701374,nm0000338	Ma Puzo,Fran Ford Coppc
on,Crime,Drama	...	Christopher Nolan	tt0468569,nm0634300,nm0634240,nm0275286,tt0468569	Writers,Jonath Nolan,Christoph Nolan,Davic

4. Visualizing Data:

```
In [18]: sns.countplot(data=train,x='imbd_rating')
Out[18]: <Axes: xlabel='imbd_rating', ylabel='count'>
```



5. Word Cloud:

```
word=WordCloud(width=200,height=200,
               background_color='white',
               min_font_size=10).generate(str(train['review_content1']))
plt.figure(figsize=(5,5),facecolor=None)
plt.imshow(word)
plt.axis('off')
plt.show()
```

6. TF-IDF:

```
d0=" the boys is playing games in criket."
d1="the boys are mostly like this games."
d2="it is one of the farvarite games."
string =[d0,d1,d2]
```

```
#Get tf-idf values from fit_transform() method.
```

```
tdf=TfidfVectorizer()
result=tdf.fit_transform(string)
```

```
import numpy as np
```

```
print(result)
```

```
  (0, 2)        0.45386826657073503
  (0, 5)        0.45386826657073503
  (0, 4)        0.2680619096684997
  (0, 12)       0.45386826657073503
  (0, 6)        0.34517851538731575
  (0, 1)        0.34517851538731575
  (0, 13)       0.2680619096684997
  (1, 14)       0.43535684236960664
  (1, 8)        0.43535684236960664
  (1, 9)        0.43535684236960664
  (1, 0)        0.43535684236960664
  (1, 4)        0.25712876433201076
  (1, 1)        0.33110010014200913
  (1, 13)       0.25712876433201076
  (2, 3)        0.43535684236960664
  (2, 10)       0.43535684236960664
  (2, 11)       0.43535684236960664
  (2, 7)        0.43535684236960664
  (2, 4)        0.25712876433201076
  (2, 6)        0.33110010014200913
  (2, 13)       0.25712876433201076
```

```
print(tdf.get_feature_names_out())
```

```
['are' 'boys' 'criket' 'farvarite' 'games' 'in' 'is' 'it' 'like' 'mostly'
 'of' 'one' 'playing' 'the' 'this']
```

```
print(np.array(tdf.idf_))
```

```
[1.69314718 1.28768207 1.69314718 1.69314718 1.         1.69314718
 1.28768207 1.69314718 1.69314718 1.69314718 1.69314718 1.69314718
 1.69314718 1.         1.69314718]
```

```
# get idf values
print('\nidf values:')
for ele1,ele2 in zip(tdf.get_feature_names_out(),tdf.idf_):
    print(ele1,':',ele2)
```

```
idf values:
are : 1.6931471805599454
boys : 1.2876820724517808
criket : 1.6931471805599454
farvarite : 1.6931471805599454
games : 1.0
in : 1.6931471805599454
is : 1.2876820724517808
it : 1.6931471805599454
like : 1.6931471805599454
mostly : 1.6931471805599454
of : 1.6931471805599454
one : 1.6931471805599454
playing : 1.6931471805599454
the : 1.0
this : 1.6931471805599454
```

In [9]:
```
print('\nword indexes:')
print(tdf.vocabulary_)
print('\ntf-idf value:')
print(result)
print('\ntf-idf values in matrix form:')
print(result.toarray())
```

```
word indexes:
{'the': 13, 'boys': 1, 'is': 6, 'playing': 12, 'games': 4, 'in': 5, 'criket': 2, 'are': 0, 'mostly':
9, 'like': 8, 'this': 14, 'it': 7, 'one': 11, 'of': 10, 'farvarite': 3}

tf-idf value:
  (0, 2)        0.45386826657073503
  (0, 5)        0.45386826657073503
  (0, 4)        0.2680619096684997
  (0, 12)       0.45386826657073503
  (0, 6)        0.34517851538731575
  (0, 1)        0.34517851538731575
  (0, 13)       0.2680619096684997
  (1, 14)       0.43535684236960664
  (1, 8)        0.43535684236960664
  (1, 9)        0.43535684236960664
  (1, 0)        0.43535684236960664
  (1, 4)        0.25712876433201076
  (1, 1)        0.33110010014200913
  (1, 13)       0.25712876433201076
  (2, 3)        0.43535684236960664
  (2, 10)       0.43535684236960664
  (2, 11)       0.43535684236960664
  (2, 7)        0.43535684236960664
  (2, 4)        0.25712876433201076
  (2, 6)        0.33110010014200913
  (2, 13)       0.25712876433201076

tf-idf values in matrix form:
[[0.         0.34517852 0.45386827 0.         0.26806191 0.45386827
  0.34517852 0.         0.         0.         0.         0.
  0.45386827 0.26806191 0.        ]
 [0.43535684 0.3311001  0.         0.         0.25712876 0.
  0.         0.         0.43535684 0.43535684 0.         0.
  0.         0.25712876 0.43535684]
 [0.         0.         0.         0.43535684 0.25712876 0.
  0.3311001  0.43535684 0.         0.         0.43535684 0.43535684
  0.         0.25712876 0.        ]]
```

RESULT:

Thus, the programs are completed successfully executed.

Ex. no: 6	Naïve Bayes Classifier
DATE:06-08-2024	

AIM:

To understand and implement the Naive Bayes Classifier for text classification tasks.

PROCEDURE:

1) Import libraries
2) Load the dataset
3) Describe the Data
4) Preprocess Data
5) Build Model

PROGRAM:

Import Libraries

```
import pandas as pd
import string
import re
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report,confusion_matr
```

Load the dataset

```
data3 = pd.read_csv(r"E:\IMDB-Dataset.csv")
```

```
data3.head()
```

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

Describe the Data

```
data3.describe()
```

	review	sentiment
count	50000	50000
unique	49582	2
top	Loved today's show!!! It was a variety and not...	positive
freq	5	25000

Preprocess Data

```
def clean_text(text):
    # Remove punctuation and lower text
    text = text.translate(str.maketrans('', '', string.punctuation)).lower()
    # Tokenize and remove stopwords
    stop_words = set(stopwords.words('english'))
    text = ' '.join([word for word in text.split() if word not in stop_words])
    return text
```

```
data3['review'] = data3['review'].map(clean_text)
```

```
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(data3['review'])
y = data3['sentiment']  # Assuming 'sentiment' is the target column
```

- Splitting Data into Training and Testing Sets

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

Build Model

```
model = MultinomialNB()
```

```
model.fit(X_train, y_train)
```

```
MultinomialNB
MultinomialNB()
```

- Finding Accuracy and Prediction

```
y_pred = model.predict(X_test)
```

```
print(f"Accuracy: {accuracy_score(y_test, y_pred)*100}")
```

```
Accuracy: 86.17
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

    negative       0.85      0.88      0.86      4961
    positive       0.88      0.84      0.86      5039

    accuracy                           0.86     10000
   macro avg       0.86      0.86      0.86     10000
weighted avg       0.86      0.86      0.86     10000
```

```
print(confusion_matrix(y_test,y_pred))
```

```
[[4362  599]
 [ 784 4255]]
```

RESULT:

Thus, the Naive Bayes Classifier was successfully implemented, program is completed successfully with 86% accuracy

Ex. no: 7	WORD CLOUD USING PYTHON
DATE:20-08-2024	

AIM:

To apply a word cloud using Python packages for visualizing text data.

PROCEDURE:

1. Importing Libraries.
2. Loading Training and Testing Texts.
3. Preprocessing.
4. Word Cloud

PROGRAM:

1. Importing Libraries:

```
import pandas as pd
import string
import re
from nltk.corpus import stopwords
from wordcloud import WordCloud
import matplotlib.pyplot as plt
```

2. Loading Dataset:

```
data4 = pd.read_csv(r"E:\IMDB-Dataset.csv")
```

```
data4.head()
```

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

3. Preprocessing:

```
def preprocess_text(text):
    # Remove punctuation and lowercase text
    text = text.translate(str.maketrans('', '', string.punctuation)).lower()
    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    text = ' '.join([word for word in text.split() if word not in stop_words])
    return text
```

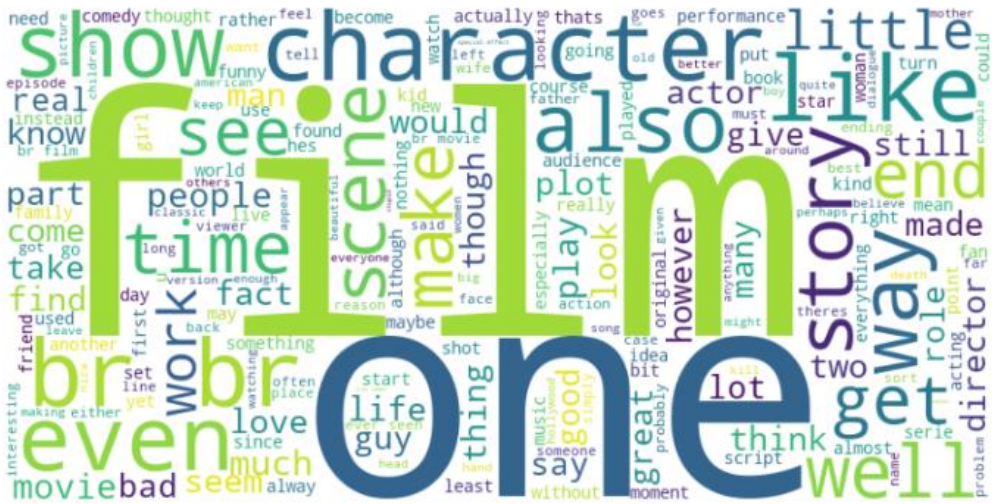
```
data3['cleaned_review'] = data3['review'].map(preprocess_text)
```

```
text = ' '.join(data3['cleaned_review'])
```

4. Word Cloud

```
wordcloud = WordCloud(width=800,
                      height=400,
                      background_color='white',
                      stopwords=stopwords.words('english')).generate(text)
```

```
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')  # Hide axes
plt.show()
```



RESULT:

Thus, a word cloud was successfully generated using Python.

Ex. no: 8	PYTHON KEYWORD EXTRACTION
DATE:27-08-2024	

AIM:

To perform keyword extraction using Python for effective text analysis.

PROCEDURE:

1. Rake
2. Yake
3. Spacy
4. Textacy

PROGRAM:

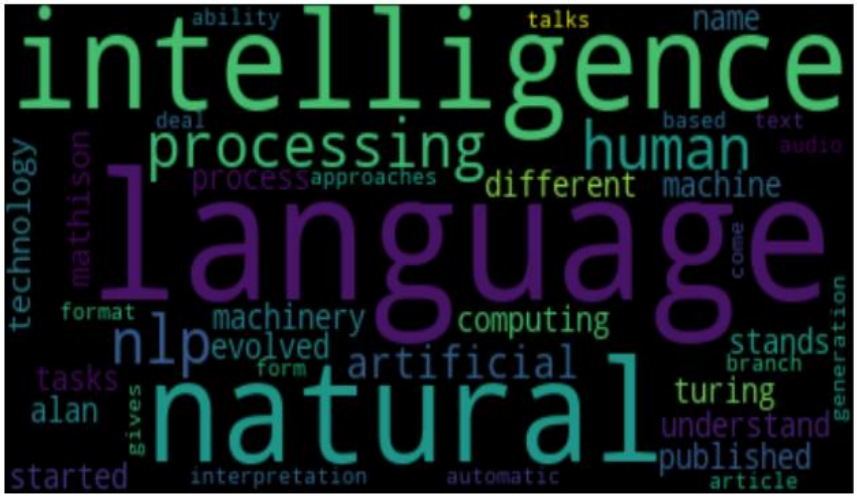
RAKE:

```
# Importing Libraries
from rake_nltk import Rake
from wordcloud import WordCloud
import matplotlib.pyplot as plt
```

```
# Initializing the Rake instance
rake = Rake()

# Input text
input_text = '''
NLP stands for Natural Language Processing.
It is the branch of Artificial Intelligence that gives the ability to machine understand
and process human languages. Human languages can be in the form of text or audio format.
Natural Language Processing started in 1950 When Alan Mathison Turing published
an article in the name Computing Machinery and Intelligence.
It is based on Artificial intelligence. It talks about automatic interpretation and
generation of natural language.
As the technology evolved, different approaches have come to deal with NLP tasks.
'''
```

```
# Extracting keywords and phrases
rake.extract_keywords_from_text(input_text)
keywords = rake.get_ranked_phrases()

# Displaying the keywords
print(keywords)

# Generate WordCloud
wordcloud = WordCloud().generate(' '.join(keywords))

# Display the WordCloud
plt.figure(figsize=(10,10))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

```
['alan mathison turing published', 'natural language processing started', 'natural language processing', 'name computing machin
ery', 'process human languages', 'natural language', 'human languages', 'technology evolved', 'nlp tasks', 'nlp stands', 'machi
ne understand', 'different approaches', 'automatic interpretation', 'audio format', 'artificial intelligence', 'artificial inte
lligence', 'intelligence', 'text', 'talks', 'gives', 'generation', 'form', 'deal', 'come', 'branch', 'based', 'article', 'abili
ty', '1950']
```

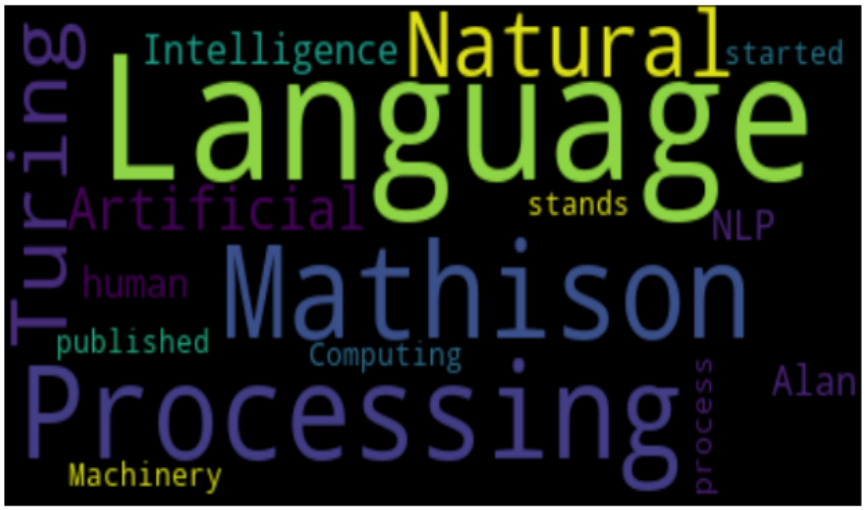


YAKE:

```
# Importing libraries
from wordcloud import WordCloud
import matplotlib.pyplot as plt
import yake

# Initializing the YAKE instance
yake_kw = yake.KeywordExtractor()

# Input text
input_text = '''
NLP stands for Natural Language Processing.
It is the branch of Artificial Intelligence that gives the ability to machine understand
and process human languages. Human languages can be in the form of text or audio format.
Natural Language Processing started in 1950 When Alan Mathison Turing published
an article in the name Computing Machinery and Intelligence.
It is based on Artificial intelligence. It talks about automatic interpretation and
generation of natural language.
As the technology evolved, different approaches have come to deal with NLP tasks.
'''
```

```
# Extracting keywords
KeyWords = yake_kw.extract_keywords(input_text)

# Displaying the keywords
print(KeyWords)

# Extracting keywords
keywords = [kw for kw, _ in KeyWords]

# Generate WordCloud
wordcloud = WordCloud().generate(' '.join(keywords))

# Display the WordCloud
plt.figure(figsize=(10, 10))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

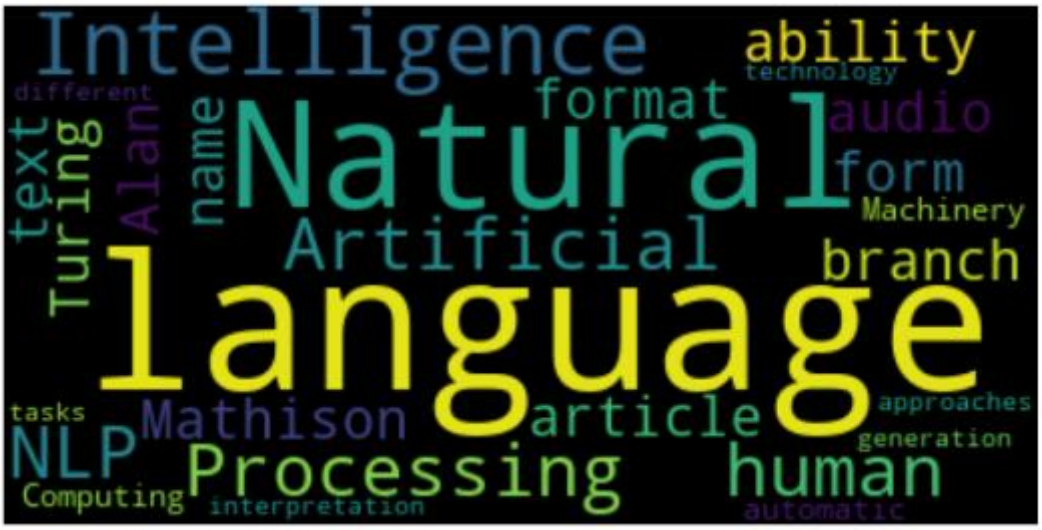
```
[('Natural Language Processing', 0.02100249013859125), ('Language Processing', 0.04163335302639552), ('Natural Language', 0.048148820863336377), ('Artificial Intelligence', 0.06657427591685054), ('Alan Mathison Turing', 0.06812525384060124), ('Language Processing started', 0.0760442529090274 7), ('human languages', 0.08215351904804695), ('NLP stands', 0.09173112596477705), ('Language', 0.10178153594306494), ('process human languages', 0.11865807800247614), ('Processing', 0.12586811799925435), ('Intelligence', 0.12825620909446891), ('Natural', 0.13778435888897436), ('Alan Mathison', 0.15153101048626974), ('Mathison Turing', 0.15153101048626974), ('Computing Machinery', 0.15153101048626974), ('Mathison Turing published', 0.15160281730925312), ('languages', 0.1526723039145974), ('Artificial', 0.15269328890550202), ('NLP', 0.18058428305612767)]
```



SPACY:

```
# Importing Libraries
import spacy
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Initializing the spaCy model instance
nlp = spacy.load('en_core_web_sm')

# Input text
input_text = '''
NLP stands for Natural Language Processing.
It is the branch of Artificial Intelligence that gives the ability to machine understand
and process human languages. Human languages can be in the form of text or audio format.
Natural Language Processing started in 1950 When Alan Mathison Turing published
an article in the name Computing Machinery and Intelligence.
It is based on Artificial intelligence. It talks about automatic interpretation and
generation of natural language.
As the technology evolved, different approaches have come to deal with NLP tasks.
'''
```

```
# Creating a spaCy document
spacy_doc = nlp(input_text)

# Initializing keywords list variable
keywords = []

# Extracting keyphrases
for chunk in spacy_doc.noun_chunks:
    if chunk.text.lower() not in nlp.Defaults.stop_words:
        keywords.append(chunk.text)

# Displaying the keywords
print(keywords)

# Generate WordCloud
wordcloud = WordCloud().generate(' '.join(keywords))

# Display the WordCloud
plt.figure(figsize=(10,10))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



TEXTACY:

```
import textacy
```

```
import textacy.datasets
from wordcloud import WordCloud
import matplotlib.pyplot as plt
```

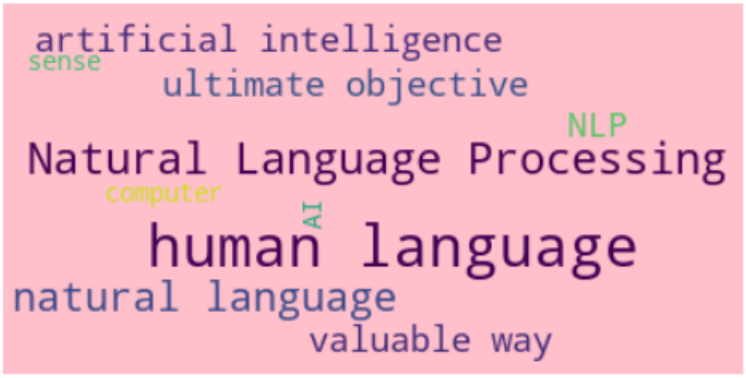
```
# Example text
text = """
Natural Language Processing (NLP) is a subfield of artificial intelligence (AI) that i
between computers and humans through natural language. The ultimate objective of NLP i
and make sense of the human languages in a valuable way.
"""
```

```
# Load spaCy's English model
nlp = spacy.load('en_core_web_sm')
```

```
C:\Users\Gowth\anaconda3\Lib\site-packages\spacy\util.py:910: UserWarning: [W095] Model 'en_core_web_sm' (3.8.0) was
trained with spaCy v3.8.0 and may not be 100% compatible with the current version (3.7.5). If you see errors or degra
ded performance, download a newer compatible model or retrain your custom model with the current spaCy version. For m
ore details and available updates, run: python -m spacy validate
  warnings.warn(warn_msg)
```

```
# Process text using textacy
doc = textacy.make_spacy_doc(text, lang=nlp)
```

```
# Extract key terms using Textacy
keywords = textacy.extract.keyterms.textrank(doc, topn=10)

# Convert keywords to a format suitable for wordcloud
word_freq = {term: score for term, score in keywords}

# Generate the word cloud with a grey background and smaller size
wordcloud = WordCloud(width=400, height=200, background_color='pink').generate_from_frequencies(word_freq)

# Display the word cloud using matplotlib
plt.figure(figsize=(6, 3))  # Smaller figure size
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')  # Turn off axis
plt.show()
```



RESULT:

The program is completed with using four different keyword extraction Techniques.

Ex. no: 9	NAMED ENTITY RECOGNITION
DATE:31-08-2024	

AIM:

To Named entity recognition in NLP

PROCEDURE:

1. Prepare Text Data
2. Process the Text
3. Extract Named Entities
4. Visualize Named Entities

PROGRAM:

- Prepare Text Data

```
# Example text
text = """
Elon Musk, the CEO of Tesla, visited Berlin last month. Tesla's Gigafactory in Texas has begun production of the new Model Y.
Apple Inc. also made announcements regarding its new product line in California.
"""
```

- Process the Text

```
# Process the text using spaCy
doc = nlp(text)
```

- Extract Named Entities

```
# Extract named entities
for entity in doc.ents:
    print(f"{entity.text}: {entity.label_}")
```

```
Elon Musk: PERSON
Tesla: ORG
Berlin: GPE
last month: DATE
Tesla: ORG
Texas: GPE
Model Y.
Apple Inc.: PERSON
California: GPE
```

- Visualize Named Entities

```
from spacy import displacy

# Visualize the named entities
displacy.render(doc, style="ent", jupyter=True)
```

Elon Musk PERSON , the CEO of Tesla ORG , visited Berlin GPE last month DATE . Tesla ORG 's Gigafactory in Texas GPE has begun production of the new Model Y. Apple Inc. PERSON also made announcements regarding its new product line in California GPE .

RESULT:

Thus, the NER (NAMED ENTITY RECOGNITION) is Successfully executed.

Ex. no: 10	LATENT SEMANTIC ANALYSIS
DATE:10-09-2024	

AIM:

To perform Latent Semantic Analysis for uncovering relationships between terms and documents in a text corpus.

PROCEDURE:

1. Import libraries
2. Load Data
3. Vectorization
4. Latent Semantic Analysis (LSA)

PROGRAM:

1. Import libraries:

```
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
```

2. Load Data:

```
documents = [
    "The cat sat on the mat",
    "The dog sat on the mat",
    "Cats and dogs are animals",
    "The mat is blue",
    "The quick brown fox jumps over the lazy dog",
    "The dog is a loyal animal",
    "Cats are independent animals",
    "The blue mat is new"
]
```

3. Vectorization:

```
vectorizer = TfidfVectorizer(stop_words="english")
```

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

4. Latent Semantic Analysis (LSA)

```
n_components = 2  # Number of latent topics
svd_model = TruncatedSVD(n_components=n_components)
lsa = svd_model.fit_transform(X)
```

```
terms = vectorizer.get_feature_names_out()
```

```
for i, comp in enumerate(svd_model.components_):
    terms_comp = zip(terms, comp)
    sorted_terms = sorted(terms_comp, key=lambda x: x[1], reverse=True)

    print(f"Concept {i + 1}:")
    for term in sorted_terms[:5]:
        print(term[0], term[1])
    print("\n")
```

```
Concept 1:
mat 0.6654749745613435
blue 0.4814579077859333
sat 0.39811489017658575
dog 0.24600273021349084
new 0.2307982761360983


Concept 2:
animals 0.6072471929778377
cats 0.6072471929778377
independent 0.36228558710020087
dogs 0.36228558710020037
blue 5.006195932023018e-16
```

```
print("Latent representation of documents:")
print(lsa)
```

```
Latent representation of documents:
[[ 6.66510918e-01 -5.26887179e-16]
 [ 7.31766736e-01 -1.06787355e-15]
 [ 3.17080752e-16  8.89987722e-01]
 [ 7.85467963e-01  2.91256890e-16]
 [ 1.29689510e-01 -2.67189687e-16]
 [ 1.76248196e-01 -9.43290839e-16]
 [ 3.02064967e-16  8.89987722e-01]
 [ 7.28122133e-01  3.07517665e-16]]
```

RESULT:

Thus, Latent semantic analysis executed and completed successfully.

EX. NO: 11	DETERMINE OPTIMUM NUMBER OF TOPICS IN A DOCUMENT
DATE: 18-09-2024	

AIM:

To determine the optimum number of topics in a document using topic modeling techniques.

PROCEDURE:

1. import libraries
2. Load the text Documents
3. Data Preprocessing
4. Create Model
5. Plotting Graph
6. Determine and Display Optimal Number of Topics

PROGRAM:

- import libraries

```
import numpy as np
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
from gensim.models import LdaModel
from gensim.corpora import Dictionary
from gensim.models.coherencemodel import CoherenceModel
import string
```

- Load the text Documents

```
docs = [
    "The cat sat on the mat.",
    "The dog sat on the mat.",
    "Cats and dogs are animals.",
    "The mat is blue.",
    "The quick brown fox jumps over the lazy dog.",
    "The dog is a loyal animal.",
    "Cats are independent animals.",
    "The blue mat is new.",
    "Artificial intelligence is revolutionizing industries.",
    "Machine learning and data science are important fields.",
    "Natural language processing is a subset of AI.",
    "Deep learning models are used for image recognition.",
    "Healthcare technology is advancing rapidly.",
    "Financial technology is transforming banking."
]
```

```
translator = str.maketrans("", "", string.punctuation)
```

- Preprocess the data

```
def preprocess(text):
    tokens = text.lower().translate(translator).split()
    return [word for word in tokens if word not in stop_words]
```

```
preprocessed_docs = [preprocess(doc) for doc in docs]
```

- Create Dictionary and Corpus

```
dictionary = Dictionary(preprocessed_docs)
corpus = [dictionary.doc2bow(doc) for doc in preprocessed_docs]
```

- Calculate Coherence Score for LDA Model

```
def calculate_coherence(dictionary, corpus, texts, num_topics):
    lda_model = LdaModel(corpus=corpus, id2word=dictionary, num_topics=num_topic
    coherence_model = CoherenceModel(model=lda_model, texts=texts, dictionary=di
    coherence_score = coherence_model.get_coherence()
    return coherence_score
```

- Define Topic Range and Initialize Coherence Scores

```
min_topics = 2
max_topics = 10
step_size = 1
topic_range = range(min_topics, max_topics + 1, step_size)
coherence_scores = []
```

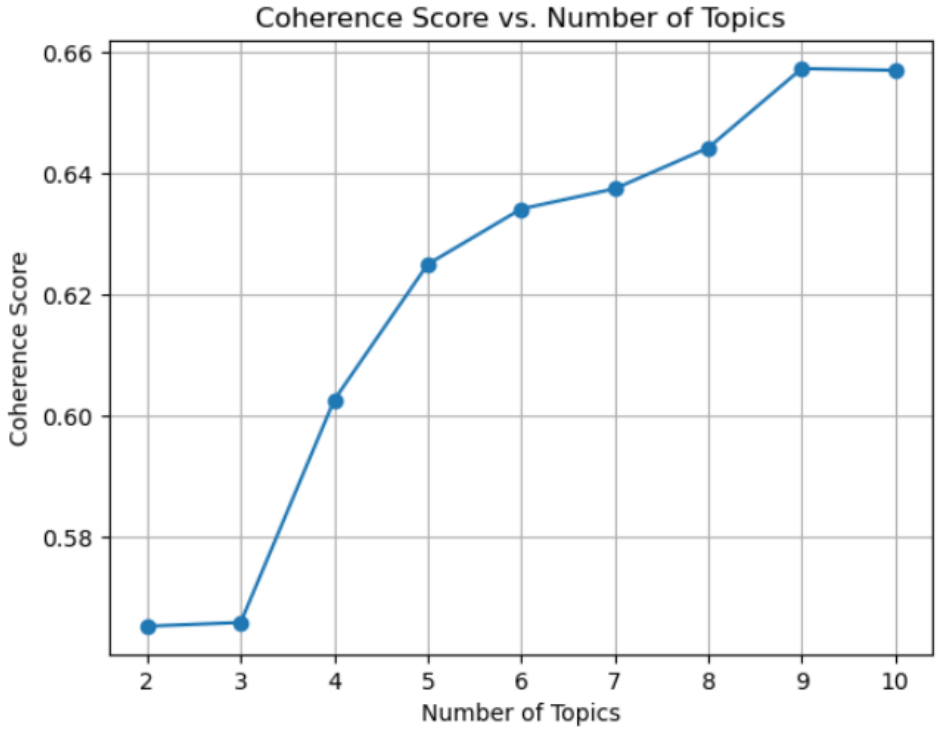
- Calculate Coherence Scores for Each Topic Count

```
for num_topics in topic_range:
    coherence_score = calculate_coherence(dictionary, corpus, preprocessed_docs,
    coherence_scores.append(coherence_score)
```

- Plot: Coherence Score vs. Number of Topics

```
plt.plot(topic_range, coherence_scores, marker='o')
plt.xlabel('Number of Topics')
plt.ylabel('Coherence Score')
plt.title('Coherence Score vs. Number of Topics')
plt.xticks(topic_range)
plt.grid(True)
plt.show()
```

- Determine and Display Optimal Number of Topics




```
optimal_num_topics = topic_range[np.argmax(coherence_scores)]
print("Optimal number of topics:", optimal_num_topics)
print("Coherence score for optimal number of topics:", coherence_scores[np.argma
```

```
Optimal number of topics: 9
Coherence score for optimal number of topics: 0.6573807132676159
```

RESULT:

Thus, the optimal number of topics in the document set was successfully determined.

EX. NO: 12	FUNDAMENTALS OF TOPIC MODELING
DATE: 25-09-2024	

AIM:

To understand and apply the fundamentals of topic modeling.

PROCEDURE:

- import the library
- Setting Parameters for Topic Modeling
- Term Frequency Vector
- Fitting Latent Dirichlet Allocation (LDA) Model
- Displaying Top Words for Each Topic

LDA (Latent Dirichlet Allocation):

- import the library

```
In [112]: import numpy as np
          import matplotlib.pyplot as plt
          from sklearn.datasets import fetch_20newsgroups
          from sklearn.feature_extraction.text import CountVectorizer
          from sklearn.decomposition import LatentDirichletAllocation
```

```
dataset = fetch_20newsgroups(shuffle=True, random_state=1, remove=("headers", "f
documents = dataset.data
```

- Setting Parameters for Topic Modeling

```
no_features = 1000
no_topics = 10
```

- Term Frequency Vector

```
tf_vectorizer = CountVectorizer(max_df=0.95,
                                min_df=2,
                                max_features=no_features,
                                stop_words='english')
tf = tf_vectorizer.fit_transform(documents)
tf_features_names= tf_vectorizer.get_feature_names_out()
```

- Fitting Latent Dirichlet Allocation (LDA) Model

```
lda = LatentDirichletAllocation(n_components=no_topics,
                                max_iter=5,
                                learning_method='online',
                                 learning_offset=50.,
                                random_state=0)
lda.fit(tf)
```

```
LatentDirichletAllocation(learning_method='online', learning_offset=50.0,
                          max_iter=5, random_state=0)
```

- Displaying Top Words for Each Topic

```
no_top_words = 10  # Define the number of top words to display

for topic_idx, topic in enumerate(lda.components_):
    print("Topic %d:" % (topic_idx))
    print(" ".join([tf_features_names[i]
                    for i in topic.argsort()[:-no_top_words - 1:-1]]))
```

```
Topic 0:
people gun armenian armenians war turkish states israel said children
Topic 1:
government people law mr use president don think right public
Topic 2:
space program output entry data nasa use science research build
Topic 3:
key car chip used keys bike use bit clipper number
Topic 4:
edu file com available mail ftp files information image send
Topic 5:
god people does jesus say think believe don know just
Topic 6:
windows use drive thanks does problem know card like using
Topic 7:
ax max b8f g9v a86 pl 145 1d9 0t 34u
Topic 8:
just don like think know good time ve people said
Topic 9:
10 00 25 15 12 20 11 14 17 16
```

RESULT:

The Fundamentals of Topic Modeling based on the implementation of Latent Dirichlet Allocation (LDA)