EX.NO:1	FUNDAMENTALS OF NLP I TOKENIZATION & LEMMATIZATION
DATE:04.07.23	

AIM:

To know the Fundamentals of NLP I Tokenization & Lemmatization.

INTRODUCTION TO TOKENIZATION:

Tokenization is the breaking of text into small chunks. Tokenization splits the text (sentence, paragraph) into words, sentences called tokens. These tokens help in interpreting the meaning of the text by analyzing the sequence of tokens.

If the text is split into sentences using some separation technique it is known as *sentence tokenization* and the same separation done for words is known as *word tokenization*.

INTRODUCTION TO LEMMATIZATION:

Lemmatization is the process of grouping together different inflected forms of the same word. It's used in computational linguistics, natural language processing (NLP) and chatbots. Lemmatization links similar meaning words as one word, making tools such as chatbots and search engine queries more effective and accurate.

The goal of lemmatization is to reduce a word to its root form, also called a *lemma*. For example, the verb "running" would be identified as "run." Lemmatization studies the morphological, or structural, and contextual analysis of words.

To correctly identify a lemma, tools analyze the context, meaning and the intended part of speech in a sentence, as well as the word within the larger context of the surrounding sentence, neighboring sentences or even the entire document. With this in-depth understanding, tools that use lemmatization can better understand the meaning of a sentence.

PROCEDURE:

Step:1 Load the necessary packages.

```
import nltk

from nltk.tokenize import word_tokenize,sent_tokenize,TreebankWordTokenizer,wordpunct_tokenize,TweetTokenizer,MWETokenizer
```

Step:2 give a sentence as input and use different type of tokenization of words.

```
a=str(input('Enter the sentance:'))
print("----------------------------------------------------------------------------------------------------------")
print("Word tokenize :",word_tokenize(a))
print('----------------------------------------------------------------------------------------------------------')
print("Sentance tokenize :",sent_tokenize(a))
print('----------------------------------------------------------------------------------------------------------')
print("Word punctuation tokenize :",wordpunct_tokenize(a))
```

```
Enter the sentance:I have three visions for India. In 3000 years of our history, people from all over the world have come and i
nvaded us.                From Alexander onwards, the Greeks, the Turks, all of them came and looted us, took over what was ou
rs.               Yet we have not done this to any other nation. We have not conquered anyone.                We have not gr
abbed their land, their culture, their history and tried to enforce our way of life on them.
----------------------------------------------------------------------------------------------------
Word tokenize : ['I', 'have', 'three', 'visions', 'for', 'India', '.', 'In', '3000', 'years', 'of', 'our', 'history', ',', 'peo
ple', 'from', 'all', 'over', 'the', 'world', 'have', 'come', 'and', 'invaded', 'us', '.', 'From', 'Alexander', 'onwards', ',',
'the', 'Greeks', ',', 'the', 'Turks', ',', 'all', 'of', 'them', 'came', 'and', 'looted', 'us', ',', 'took', 'over', 'what', 'wa
s', 'ours', '.', 'Yet', 'we', 'have', 'not', 'done', 'this', 'to', 'any', 'other', 'nation', '.', 'We', 'have', 'not', 'conquer
ed', 'anyone', '.', 'We', 'have', 'not', 'grabbed', 'their', 'land', ',', 'their', 'culture', ',', 'their', 'history', 'and',
'tried', 'to', 'enforce', 'our', 'way', 'of', 'life', 'on', 'them', '.']
----------------------------------------------------------------------------------------------------
Sentance tokenize : ['I have three visions for India.', 'In 3000 years of our history, people from all over the world have come
and invaded us.', 'From Alexander onwards, the Greeks, the Turks, all of them came and looted us, took over what was ours.', 'Y
et we have not done this to any other nation.', 'We have not conquered anyone.', 'We have not grabbed their land, their cultur
e, their history and tried to enforce our way of life on them.']
----------------------------------------------------------------------------------------------------
Word punctuation tokenize : ['I', 'have', 'three', 'visions', 'for', 'India', '.', 'In', '3000', 'years', 'of', 'our', 'histor
y', ',', 'people', 'from', 'all', 'over', 'the', 'world', 'have', 'come', 'and', 'invaded', 'us', '.', 'From', 'Alexander', 'on
wards', ',', 'the', 'Greeks', ',', 'the', 'Turks', ',', 'all', 'of', 'them', 'came', 'and', 'looted', 'us', ',', 'took', 'ove
r', 'what', 'was', 'ours', '.', 'Yet', 'we', 'have', 'not', 'done', 'this', 'to', 'any', 'other', 'nation', '.', 'We', 'have',
'not', 'conquered', 'anyone', '.', 'We', 'have', 'not', 'grabbed', 'their', 'land', ',', 'their', 'culture', ',', 'their', 'his
tory', 'and', 'tried', 'to', 'enforce', 'our', 'way', 'of', 'life', 'on', 'them', '.']
```

```
b=str(input("Enter the sentance :"))
print('----------------------------------------------------------------------------------------------------------')
print('Tree bank tokenizer :',TreebankWordTokenizer().tokenize(b))
print('----------------------------------------------------------------------------------------------------------')
print('Tweet tokenizer :',TweetTokenizer().tokenize(b))
print('----------------------------------------------------------------------------------------------------------')
print('Multi word extraction tokenizer :',MWETokenizer().tokenize(b))
```

```
Enter the sentance :I have three visions for India. In 3000 years of our history, people from all over the world have come and
invaded us.                From Alexander onwards, the Greeks, the Turks, all of them came and looted us, took over what was o
urs.               Yet we have not done this to any other nation. We have not conquered anyone.                We have not g
rabbed their land, their culture, their history and tried to enforce our way of life on them.
----------------------------------------------------------------------------------------------------
Tree bank tokenizer : ['I', 'have', 'three', 'visions', 'for', 'India.', 'In', '3000', 'years', 'of', 'our', 'history', ',', 'p
eople', 'from', 'all', 'over', 'the', 'world', 'have', 'come', 'and', 'invaded', 'us.', 'From', 'Alexander', 'onwards', ',', 't
he', 'Greeks', ',', 'the', 'Turks', ',', 'all', 'of', 'them', 'came', 'and', 'looted', 'us', ',', 'took', 'over', 'what', 'wa
s', 'ours.', 'Yet', 'we', 'have', 'not', 'done', 'this', 'to', 'any', 'other', 'nation.', 'We', 'have', 'not', 'conquered', 'an
yone.', 'We', 'have', 'not', 'grabbed', 'their', 'land', ',', 'their', 'culture', ',', 'their', 'history', 'and', 'tried', 't
o', 'enforce', 'our', 'way', 'of', 'life', 'on', 'them', '.']
----------------------------------------------------------------------------------------------------
Tweet tokenizer : ['I', 'have', 'three', 'visions', 'for', 'India', '.', 'In', '3000', 'years', 'of', 'our', 'history', ',', 'p
eople', 'from', 'all', 'over', 'the', 'world', 'have', 'come', 'and', 'invaded', 'us', '.', 'From', 'Alexander', 'onwards',
',', 'the', 'Greeks', ',', 'the', 'Turks', ',', 'all', 'of', 'them', 'came', 'and', 'looted', 'us', ',', 'took', 'over', 'wha
t', 'was', 'ours', '.', 'Yet', 'we', 'have', 'not', 'done', 'this', 'to', 'any', 'other', 'nation', '.', 'We', 'have', 'not',
'conquered', 'anyone', '.', 'We', 'have', 'not', 'grabbed', 'their', 'land', ',', 'their', 'culture', ',', 'their', 'history',
'and', 'tried', 'to', 'enforce', 'our', 'way', 'of', 'life', 'on', 'them', '.']
----------------------------------------------------------------------------------------------------
Multi word extraction tokenizer : ['I', ' ', 'h', 'a', 'v', 'e', ' ', 't', 'h', 'r', 'e', 'e', ' ', 'v', 'i', 's', 'i', 'o',
'n', 's', ' ', 'f', 'o', 'r', ' ', 'I', 'n', 'd', 'i', 'a', '.', ' ', 'I', 'n', ' ', '3', '0', '0', '0', ' ', 'y', 'e', 'a',
'r', 's', ' ', 'o', 'f', ' ', 'o', 'u', 'r', ' ', 'h', 'i', 's', 't', 'o', 'r', 'y', ',', ' ', 'p', 'e', 'o', 'p', 'l', 'e', '
', 'f', 'r', 'o', 'm', ' ', 'a', 'l', 'l', ' ', 'o', 'v', 'e', 'r', ' ', 't', 'h', 'e', ' ', 'w', 'o', 'r', 'l', 'd', ' ', 'h',
'a', 'v', 'e', ' ', 'c', 'o', 'm', 'e', ' ', 'a', 'n', 'd', ' ', 'i', 'n', 'v', 'a', 'd', 'e', 'd', ' ', 'u', 's', '.', ' ', '
', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'F', 'r', 'o', 'm', ' ', 'A', 'l', 'e', 'x', 'a',
'n', 'd', 'e', 'r', ' ', 'o', 'n', 'w', 'a', 'r', 'd', 's', ',', ' ', 't', 'h', 'e', ' ', 'G', 'r', 'e', 'e', 'k', 's', ',', '
', 't', 'h', 'e', ' ', 'T', 'u', 'r', 'k', 's', ',', ' ', 'a', 'l', 'l', ' ', 'o', 'f', ' ', 't', 'h', 'e', 'm', ' ', 'c', 'a',
'm', 'e', ' ', 'a', 'n', 'd', ' ', 'l', 'o', 'o', 't', 'e', 'd', ' ', 'u', 's', ',', ' ', 't', 'o', 'o', 'k', ' ', 'o', 'v',
'e', 'r', ' ', 'w', 'h', 'a', 't', ' ', 'w', 'a', 's', ' ', 'o', 'u', 'r', 's', '.', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '
', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'Y', 'e', 't', ' ', 'w', 'e', ' ', 'h', 'a', 'v', 'e', ' ', 'n', 'o', 't', ' ', 'd',
'o', 'n', 'e', ' ', 't', 'h', 'i', 's', ' ', 't', 'o', ' ', 'a', 'n', 'y', ' ', 'o', 't', 'h', 'e', 'r', ' ', 'n', 'a', 't',
'i', 'o', 'n', '.', ' ', 'W', 'e', ' ', 'h', 'a', 'v', 'e', ' ', 'n', 'o', 't', ' ', 'c', 'o', 'n', 'q', 'u', 'e', 'r', 'e',
'd', ' ', 'a', 'n', 'y', 'o', 'n', 'e', '.', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '
', 'W', 'e', ' ', 'h', 'a', 'v', 'e', ' ', 'n', 'o', 't', ' ', 'g', 'r', 'a', 'b', 'b', 'e', 'd', ' ', 't', 'h', 'e', 'i', 'r',
' ', 'l', 'a', 'n', 'd', ',', ' ', 't', 'h', 'e', 'i', 'r', ' ', 'c', 'u', 'l', 't', 'u', 'r', 'e', ',', ' ', 't', 'h', 'e',
'i', 'r', ' ', 'h', 'i', 's', 't', 'o', 'r', 'y', ' ', 'a', 'n', 'd', ' ', 't', 'r', 'i', 'e', 'd', ' ', 't', 'o', ' ', 'e',
'n', 'f', 'o', 'r', 'c', 'e', ' ', 'o', 'u', 'r', ' ', 'w', 'a', 'y', ' ', 'o', 'f', ' ', 'l', 'i', 'f', 'e', ' ', 'o', 'n', '
', 't', 'h', 'e', 'm', '.', ' ']
```

PROCEDURE FOR LEMMATIZATION:

Step1: load the package

```
#wordnetLemmatizer
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
```

Step 2: do the lemmatization for the given sentence.

```
a=str(input("Enter the sentance:"))
word=word_tokenize(a)
lem=WordNetLemmatizer()
for i in word:
    print(i,':',lem.lemmatize(i))
```

```
Enter the sentance:The difference between stemming and lemmatization is, lemmatization considers the context and converts the w
ord to its meaningful base form, whereas stemming just removes the last few characters, often leading to incorrect meanings and
spelling errors.
The : The
difference : difference
between : between
stemming : stemming
and : and
lemmatization : lemmatization
is : is
, : ,
lemmatization : lemmatization
considers : considers
the : the
context : context
and : and
converts : convert
the : the
word : word
to : to
its : it
meaningful : meaningful
base : base
form : form
, : ,
whereas : whereas
stemming : stemming
just : just
removes : remove
the : the
last : last
few : few
characters : character
, : ,
often : often
leading : leading
to : to
incorrect : incorrect
meanings : meaning
and : and
spelling : spelling
errors : error
. : .
```

```
print(' '.join([lem.lemmatize(i) for i in word]))
```

```
The difference between stemming and lemmatization is , lemmatization considers the context and convert the word to it meaningfu
l base form , whereas stemming just remove the last few character , often leading to incorrect meaning and spelling error .
```

RESULT:

Thus, we did the tokenization and lemmatization for the given paragraph.

EX.NO:2	FUNDAMENTALS OF NLP II STEMMING & SENTENCE SEGMENTATION
DATE:18.07.23	

AIM:

To know the Fundamentals of NLP II Stemming & Sentence Segmentation.

INTRODUCTION TO STEMMING:

Stemming is a natural language processing technique that is used to reduce words to their base form, also known as the root form. The process of stemming is used to normalize text and make it easier to process. It is an important step in text pre-processing, and it is commonly used in information retrieval and text mining applications.

There are several different algorithms for stemming, including the Porter stemmer, Snowball stemmer, and the Lancaster stemmer. The Porter stemmer is the most widely used algorithm, and it is based on a set of heuristics that are used to remove common suffixes from words. The Snowball stemmer is a more advanced algorithm that is based on the Porter stemmer, but it also supports several other languages in addition to English. The Lancaster stemmer is a more aggressive stemmer and it is less accurate than the Porter stemmer and Snowball stemmer.

Stemming can be useful for several natural language processing tasks such as text classification, information retrieval, and text summarization. However, stemming can also have some negative effects such as reducing the readability of the text, and it may not always produce the correct root form of a word.

INTRODUCTION TO SENTENCE SEGMENTATION:

The process of deciding from where the sentences actually start or end in NLP or we can simply say that here we are dividing a paragraph based on sentences. This process is know as sentence segmentation.

PROCEDURE:

Step 1: import the packages for stemming

```
import nltk
from nltk.stem import PorterStemmer
from nltk.stem import LancasterStemmer#less accuracy compare with porter stemmer and snowball stemmer
from nltk.stem import RegexpStemmer
from nltk.stem import SnowballStemmer# advanced stemmer based on porter stemmer
```

Step 2: Do the stemming for the input

Poster stemmer:

```
#Poster stemmer
a=str(input("Enter the sentance :"))
from nltk.tokenize import word_tokenize
stemmer=PorterStemmer()

words=word_tokenize(a)
for w in words:

    print(w, ':', stemmer.stem(w))
```

```
Enter the sentance :We are importing the PorterStemmer() from the NLTK library in python in the above code. This module will he
lp in removing the suffixes of known English words.
We : we
are : are
importing : import
the : the
PorterStemmer : porterstemm
( : (
) : )
from : from
the : the
NLTK : nltk
library : librari
in : in
python : python
in : in
the : the
above : abov
code : code
. : .
This : thi
module : modul
will : will
help : help
in : in
removing : remov
the : the
suffixes : suffix
of : of
known : known
English : english
words : word
. : .
```

Lancaster stemmer:

```
#Lancaster stemmer
a=str(input("Enter the sentance :"))
from nltk.tokenize import word_tokenize
stemmer=LancasterStemmer()

words=word_tokenize(a)
for w in words:

    print(w, ':', stemmer.stem(w))
```

```
Enter the sentance :This short introduction uses Keras to: Load a prebuilt dataset. Build a neural network machine learning mod
el that classifies images.
This : thi
short : short
introduction : introduc
uses : us
Keras : kera
to : to
: : :
Load : load
a : a
prebuilt : prebuilt
dataset : dataset
. : .
Build : build
a : a
neural : neur
network : network
machine : machin
learning : learn
model : model
that : that
classifies : class
images : im
. : .
```

Regular expression stemmer:

```
#Regular expression stemmer
a=str(input("Enter the sentance :"))
from nltk.tokenize import word_tokenize
stemmer=RegexpStemmer('ing$|s$|ed$',min=2)

words=word_tokenize(a)
for w in words:

    print(w, ':', stemmer.stem(w))
```

```
Enter the sentance :This short introduction uses Keras to: Load a prebuilt dataset. Build a neural network machine learning mod
el that classifies images.
This : Thi
short : short
introduction : introduction
uses : use
Keras : Kera
to : to
: : :
Load : Load
a : a
prebuilt : prebuilt
dataset : dataset
. : .
Build : Build
a : a
neural : neural
network : network
machine : machine
learning : learn
model : model
that : that
classifies : classifie
images : image
. : .
```

Snowball stemmer:

```
#snowball stemmer
a=str(input("Enter the sentance :"))
from nltk.tokenize import word_tokenize
stemmer=SnowballStemmer('english')

words=word_tokenize(a)
for w in words:

    print(w, ':', stemmer.stem(w))
```

```
Enter the sentance :This short introduction uses Keras to: Load a prebuilt dataset. Build a neural network machine learning mod
el that classifies images.
This : this
short : short
introduction : introduct
uses : use
Keras : kera
to : to
: : :
Load : load
a : a
prebuilt : prebuilt
dataset : dataset
. : .
Build : build
a : a
neural : neural
network : network
machine : machin
learning : learn
model : model
that : that
classifies : classifi
images : imag
. : .
```

PROCEDURE FOR SENTENCE SEGMENTATION:

Step1: import the necessary packages

```
from nltk.tokenize import sent_tokenize
from nltk.corpus import  gutenberg
```

Step 2: load the data for sentence segmentation

```
#Load the data
sample=gutenberg.raw()
```

```
sample
```

```
'[Emma by Jane Austen 1816]\n\nVOLUME I\n\nCHAPTER I\n\n\nEmma Woodhouse, handsome, clever, and rich, with a comfortable hom
e\nand happy disposition, seemed to unite some of the best blessings\nof existence; and had lived nearly twenty-one years in
the world\nwith very little to distress or vex her.\n\nShe was the youngest of the two daughters of a most affectionate,\nin
dulgent father; and had, in consequence of her sister\'s marriage,\nbeen mistress of his house from a very early period.  He
r mother\nhad died too long ago for her to have more than an indistinct\nremembrance of her caresses; and her place had been
supplied\nby an excellent woman as governess, who had fallen little short\nof a mother in affection.\n\nSixteen years had Mi
ss Taylor been in Mr. Woodhouse\'s family,\nless as a governess than a friend, very fond of both daughters,\nbut particularl
y of Emma.  Between _them_ it was more the intimacy\nof sisters.  Even before Miss Taylor had ceased to hold the nominal\nof
fice of governess, the mildness of her temper had hardly allowed\nher to impose any restraint; and the shadow of authority b
eing\nnow long passed away, they had been living together as friend and\nfriend very mutually attached, and Emma doing just
what she liked;\nhighly esteeming Miss Taylor\'s judgment, but directed chiefly by\nher own.\n\nThe real evils, indeed, of E
mma\'s situation were the power of having\nrather too much her own way, and a disposition to think a little\ntoo well of her
self; these were the disadvantages which threatened\nalloy to her many enjoyments.  The danger, however, was at present\nso
unperceived, that they did not by any means rank as misfortunes\nwith her.\n\nSorrow came--a gentle sorrow--but not at all i
n the shape of any\ndisagreeable consciousness.--Miss Taylor married.  It was Miss\nTaylor\'s loss which first brought grie
f.  It was on the wedding-day\nof this beloved friend that Emma first sat in mournful thought\nof any continuance.  The wedd
ing over, and the bride-people gone,\nher father and herself were left to dine together, with no prospect\nof a third to che
er a long evening.  Her father composed himself\nto sleep after dinner, as usual, and she had then only to sit\nand think of
what she had lost.\n\nThe event had every promise of happiness for her friend.  Mr. Weston\nwas a man of unexceptionable cha
```

Step 3: Sentence segmentation.

```
#sentence segmmentation
sent=sent_tokenize(sample)
```

```
sent
```

```
['[Emma by Jane Austen 1816]\n\nVOLUME I\n\nCHAPTER I\n\n\nEmma Woodhouse, handsome, clever, and rich, with a comfortable ho
me\nand happy disposition, seemed to unite some of the best blessings\nof existence; and had lived nearly twenty-one years i
n the world\nwith very little to distress or vex her.',
 "She was the youngest of the two daughters of a most affectionate,\nindulgent father; and had, in consequence of her siste
r's marriage,\nbeen mistress of his house from a very early period.",
 'Her mother\nhad died too long ago for her to have more than an indistinct\nremembrance of her caresses; and her place had
been supplied\nby an excellent woman as governess, who had fallen little short\nof a mother in affection.',
 "Sixteen years had Miss Taylor been in Mr. Woodhouse's family,\nless as a governess than a friend, very fond of both daught
ers,\nbut particularly of Emma.",
 'Between _them_ it was more the intimacy\nof sisters.',
 "Even before Miss Taylor had ceased to hold the nominal\noffice of governess, the mildness of her temper had hardly allowed
\nher to impose any restraint; and the shadow of authority being\nnow long passed away, they had been living together as fri
end and\nfriend very mutually attached, and Emma doing just what she liked;\nhighly esteeming Miss Taylor's judgment, but di
rected chiefly by\nher own.",
 "The real evils, indeed, of Emma's situation were the power of having\nrather too much her own way, and a disposition to th
ink a little\ntoo well of herself; these were the disadvantages which threatened\nalloy to her many enjoyments.",
 'The danger, however, was at present\nso unperceived, that they did not by any means rank as misfortunes\nwith her.',
 'Sorrow came--a gentle sorrow--but not at all in the shape of any\ndisagreeable consciousness.--Miss Taylor married.',
 "It was Miss\nTaylor's loss which first brought grief.",
```

RESULT:

Thus, we did the different type of stemming and sentence segmentation.

EX.NO: 3	
DATE:25.07.23	NLP USING SCIKIT LEARN

AIM:

To know about how to use scikit learn for NLP.

INTRODUCTION TO SCIKIT LEARN:

The goal of this guide is to explore some of the main scikit-learn tools on a single practical task: analyzing a collection of text documents (movie review data).

In this section we will see how to:

- load the file contents and the categories
- extract feature vectors suitable for machine learning
- train a linear model to perform categorization
- use a grid search strategy to find a good configuration of both the feature extraction components and the classifier.

PROCEDURE:

Step 1: Import the necessary library.

```
import pandas as pd
import numpy as np
```

Step 2: load the dataset

```
df=pd.read_csv(r"C:\Users\jeevi\Downloads\movie.csv")
```

```
df.head()
```

	0	1
0	This film is absolutely awful, but nevertheles...	0
1	Well since seeing part's 1 through 3 I can hon...	0
2	I got to see this film at a preview and was da...	1
3	This adaptation positively butchers a classic ...	0
4	Råzone is an awful movie! It is so simple. It ...	0

Step 3: Information about the dataset

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 0 to 24999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   0       25000 non-null  object
 1   1       25000 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 390.8+ KB
```

Step 4: Import the count vectorizer to create the bag of words.

```
from sklearn.feature_extraction.text import CountVectorizer
count_vec = CountVectorizer()
x = count_vec.fit_transform(df['0'])
x
```

```
<25000x74849 sparse matrix of type '<class 'numpy.int64'>'
        with 3445861 stored elements in Compressed Sparse Row format>
```

Step 5: Assign x and y

```
x = count_vec.fit_transform(df['0'])
x
```

```
<25000x74849 sparse matrix of type '<class 'numpy.int64'>'
        with 3445861 stored elements in Compressed Sparse Row format>
```

```
y=df['1']
y
```

```
0        0
1        0
2        1
3        0
4        0
        ..
24995    1
24996    1
24997    1
24998    1
24999    1
Name: 1, Length: 25000, dtype: int64
```

```
x.shape
```

```
(25000, 74849)
```

```
type(x)
```

```
scipy.sparse._csr.csr_matrix
```

Step 6: Split the data into train and test

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y)
```

Step 7: build the model and train the model

```
from sklearn.naive_bayes import MultinomialNB
mnb=MultinomialNB()
mnb.fit(x_train,y_train)
```

```
MultinomialNB()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Step: 8 predict the model performance

```
predict=mnb.predict(x_test)
```

Step:9 accuracy of the model

```
from sklearn.metrics import classification_report
print(classification_report(y_test,predict))
```

```
              precision    recall  f1-score   support

           0       0.84      0.88      0.86      3139
           1       0.87      0.83      0.85      3111

    accuracy                           0.85      6250
   macro avg       0.85      0.85      0.85      6250
weighted avg       0.85      0.85      0.85      6250
```

RESULT:

Thus, we used scikit learn library for nlp.

EX.NO:4	NLP USING SPACY LIBRARY
DATE:01.08.23	

AIM:

To know how to use spacy library for NLP.

PROCEDURE:

Step 1: Import the necessary packages.

```
import spacy
from collections import Counter
import string
from nltk.corpus import stopwords
```

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

Step 2: Using counter function to the spacy input.

```
#give the spacy input
d=nlp(input('Enter the snetance:'))
counter=Counter(d)
print(counter)
```

```
Enter the snetance:Locate the configuration file in your file system. By defaul
t, it should be located in the following directory: ~/.jupyter/jupyter_notebook
_config.py
Counter({Locate: 1, the: 1, configuration: 1, file: 1, in: 1, your: 1, file: 1,
system: 1, .: 1, By: 1, default: 1, ,: 1, it: 1, should: 1, be: 1, located: 1,
in: 1, the: 1, following: 1, directory: 1, :: 1, ~/.jupyter: 1, /: 1, jupyter_n
otebook_config.py: 1})
```

Counter is an unordered collection where elements are stored as Dict keys and their count as dict value. Counter elements count can be positive, zero or negative integers. However there is no restriction on it's keys and values. Although values are intended to be numbers but we can store other objects too.

Step 3: Do the most common in frequency.

```
fre=[token.text for token in d]
```

```
freq=Counter(fre)
```

```
e=freq.most_common
```

```
e
```

```
<bound method Counter.most_common of Counter({'the': 2, 'file': 2, 'in': 2, 'Lo
cate': 1, 'configuration': 1, 'your': 1, 'system': 1, '.': 1, 'By': 1, 'defaul
t': 1, ',': 1, 'it': 1, 'should': 1, 'be': 1, 'located': 1, 'following': 1, 'di
rectory': 1, ':': 1, '~/.jupyter': 1, '/': 1, 'jupyter_notebook_config.py': 1})
>
```

```
type(e)
```

```
method
```

Step 4: print the tag, parts of speech and explanation of tags for the spacy input

```
d=nlp(input('Enter the snetance:'))
for token in d:
    print(token,token.tag_,token.pos_,spacy.explain(token.tag_))
```

```
Locate VB VERB verb, base form
the DT DET determiner
configuration NN NOUN noun, singular or mass
file NN NOUN noun, singular or mass
in IN ADP conjunction, subordinating or preposition
your PRP$ PRON pronoun, possessive
file NN NOUN noun, singular or mass
system NN NOUN noun, singular or mass
. . PUNCT punctuation mark, sentence closer
By IN ADP conjunction, subordinating or preposition
default NN NOUN noun, singular or mass
, , PUNCT punctuation mark, comma
it PRP PRON pronoun, personal
should MD AUX verb, modal auxiliary
be VB AUX verb, base form
located VBN VERB verb, past participle
in IN ADP conjunction, subordinating or preposition
the DT DET determiner
following JJ ADJ adjective (English), other noun-modifier (Chinese)
directory NN NOUN noun, singular or mass
: : PUNCT punctuation mark, colon or ellipsis
~/.jupyter NNP PROPN noun, proper singular
/ SYM SYM symbol
jupyter_notebook_config.py NN NOUN noun, singular or mass
```

RESULT:

Thus, we used spacy library for NLP.

EX.NO:5	WORKING WITH TF-IDF
DATE:08.08.23	

AIM:

To know how to work with TF-IDF.

INTRODUCTION TO TF-IDF:

TF-IDF stands for "Term Frequency – Inverse Document Frequency." It reflects how important a word is to a document in a collection or corpus. This technique is often used in information retrieval and text mining as a weighing factor.

TF-IDF is composed of two terms:

- Term Frequency (TF):
  The number of times a word appears in a document divided by the total number of words in that document.

$$tf_{i,j} = \frac{Number\ of\ times\ term\ i\ appears\ in\ document\ j}{Total\ number\ of\ terms\ in\ document\ j}$$

- Inverse Document Frequency (IDF):
  The logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

$$idf_i = log\left(\frac{Total\ number\ of\ documents}{Number\ of\ documents\ with\ term\ i\ in\ it}\right)$$

So, essentially, the TF-IDF value increases as the word's frequency in a document (TF) increases. However, this is offset by the number of times the word appears in the entire collection of documents or corpus (IDF).

PROCEDURE:

Step 1: import the necessary packages.

```
import numpy as np
import pandas as pd
```

Step 2: load the dataset

```
df=pd.read_csv(r"C:\Users\jeevi\Downloads\movie.csv")
```

```
df.head()
```

	0	1
0	This film is absolutely awful, but nevertheles...	0
1	Well since seeing part's 1 through 3 I can hon...	0
2	I got to see this film at a preview and was da...	1
3	This adaptation positively butchers a classic ...	0
4	Råzone is an awful movie! It is so simple. It ...	0

Step 3: Information and descriptive statistics about the dataset

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 0 to 24999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   0       25000 non-null  object
 1   1       25000 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 390.8+ KB
```

```
df.describe()
```

	1
count	25000.00000
mean	0.50000
std	0.50001
min	0.00000
25%	0.00000
50%	0.50000
75%	1.00000
max	1.00000

Step 4: load the TF-IDF and do vectorizer

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vec=TfidfVectorizer()
x=tfidf_vec.fit_transform(df['0'])
```

```
type(x)
```

```
scipy.sparse._csr.csr_matrix
```

Step 5: Assign x and y , split train and test dataset

```
y=df['1']
y
```

```
0        0
1        0
2        1
3        0
4        0
        ..
24995    1
24996    1
24997    1
24998    1
24999    1
Name: 1, Length: 25000, dtype: int64
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y)
```

```
x_train
```

```
<18750x74849 sparse matrix of type '<class 'numpy.float64'>'
        with 2582537 stored elements in Compressed Sparse Row format>
```

```
y_train
```

```
20287    1
14953    1
9478     0
14866    1
20446    0
        ..
14226    0
21772    1
12438    1
23565    1
2191     1
Name: 1, Length: 18750, dtype: int64
```

Step 6: build the model

```
from sklearn.naive_bayes import MultinomialNB
mnb=MultinomialNB()
mnb.fit(x_train,y_train)
```

```
MultinomialNB()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Step 7: predict the model

```
predict=mnb.predict(x_test)
```

Step 8: Accuracy of the model

```
from sklearn.metrics import classification_report
print(classification_report(y_test,predict))
```

```
              precision    recall  f1-score   support

           0       0.86      0.89      0.87      3159
           1       0.88      0.85      0.87      3091

    accuracy                           0.87      6250
   macro avg       0.87      0.87      0.87      6250
weighted avg       0.87      0.87      0.87      6250
```

RESULT:

Thus, we worked with TF-IDF.

EX.NO:6	NAÏVE BAYES CLASSIFIER
DATE:29.08.23	

AIM:

To build the model in naïve bayes classifier.

INTRODUCTION TO NAÏVE BAYES:

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable.

Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods. The decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one dimensional distribution. This in turn helps to alleviate problems stemming from the curse of dimensionality.

MULTINOMIAL NAÏVE BAYES:

MultinomialNB implements the naive Bayes algorithm for multinomially distributed data, and is one of the two classic naive Bayes variants used in text classification (where the data are typically represented as word vector counts, although tf-idf vectors are also known to work well in practice).

PROCEDURE:

Step 1: import the necessary packages.

```
import numpy as np
import pandas as pd
```

Step 2: load the dataset

```
df=pd.read_csv(r"C:\Users\jeevi\Downloads\movie.csv")
```

```
df.head()
```

	0	1
0	This film is absolutely awful, but nevertheles...	0
1	Well since seeing part's 1 through 3 I can hon...	0
2	I got to see this film at a preview and was da...	1
3	This adaptation positively butchers a classic ...	0
4	Råzone is an awful movie! It is so simple. It ...	0

Step 3: Information and descriptive statistics about the dataset

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 0 to 24999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   0       25000 non-null  object
 1   1       25000 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 390.8+ KB
```

```
df.describe()
```

	1
count	25000.00000
mean	0.50000
std	0.50001
min	0.00000
25%	0.00000
50%	0.50000
75%	1.00000
max	1.00000

Step 4: Remove punctuation and stopwords.

```
def clean(text):
    r_punt=[char for char in text if char not in string.punctuation]
    join=''.join(r_punt)
    r_s=[i for i in join.split() if i.lower() not in stopwords.words('english')]
    sent=' '.join(r_s)
    return sent
```

```
df['0']=df['0'].apply(clean)
```

```
df.head()
```

	0	1
0	film absolutely awful nevertheless hilarious t...	0
1	Well since seeing parts 1 3 honestly say NEVER...	0
2	got see film preview dazzled typical romantic ...	1
3	adaptation positively butchers classic beloved...	0
4	Råzone awful movie simple seems tried make mov...	0

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 0 to 24999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   0       25000 non-null  object
 1   1       25000 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 390.8+ KB
```

Step 4: load the count vectorizer and do vectorizer

```
from sklearn.feature_extraction.text import CountVectorizer
count_vec = CountVectorizer()
bow = count_vec.fit_transform(df['0'])
```

Step 5: Assign x and y , split train and test dataset

```
x=bow
y=df['1']
```

Step 6: build the Multinomial NB model

```
from sklearn.naive_bayes import MultinomialNB
mnb=MultinomialNB()
mnb.fit(x_train,y_train)
```

```
MultinomialNB()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Step 7: predict the model

```
predict=mnb.predict(x_test)
```

Step 8: Accuracy of the model

```
from sklearn.metrics import confusion_matrix,classification_report
print(classification_report(y_test,predict))
```

```
              precision    recall  f1-score   support

           0       0.83      0.89      0.86      3079
           1       0.88      0.83      0.85      3171

    accuracy                           0.86      6250
   macro avg       0.86      0.86      0.86      6250
weighted avg       0.86      0.86      0.86      6250
```

```
print(confusion_matrix(y_test,predict))
```

```
[[2734  345]
 [ 550 2621]]
```

RESULT:

Thus, we build the multinomial NB model for movie review dataset and got the accuracy of 86%.

EX.NO:7	WORD CLOUD USING PYTHON
DATE:05.09.23	

AIM:

To do word cloud using python packages.

INTRODUCTION TO WORD CLOUD:

The word cloud function from word cloud allows creating word clouds in Python. The function provides several methods, but generate is the one you need to create a word cloud **from a text string**. Note that by default, the image size is 400x200 but you can customize the size with width and height, as in the example below or using scale (defaults to 1), which is recommended for large word clouds.

PROCEDURE:

Step 1: import the necessary packages.

```
# importing all necessary modules
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
import pandas as pd
```

Step2: load the dataset

```
# Read file
df = pd.read_csv(r"C:\Users\jeevi\Downloads\movie.csv")
```

Step 3: create the for loop for the preprocessing

```
comment_words = ''
stopwords = set(STOPWORDS)

# iterate through the csv file
for val in df['0']:

    # typecaste each val to string
    val = str(val)

    # split the value
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    comment_words += " ".join(tokens)+" "
```

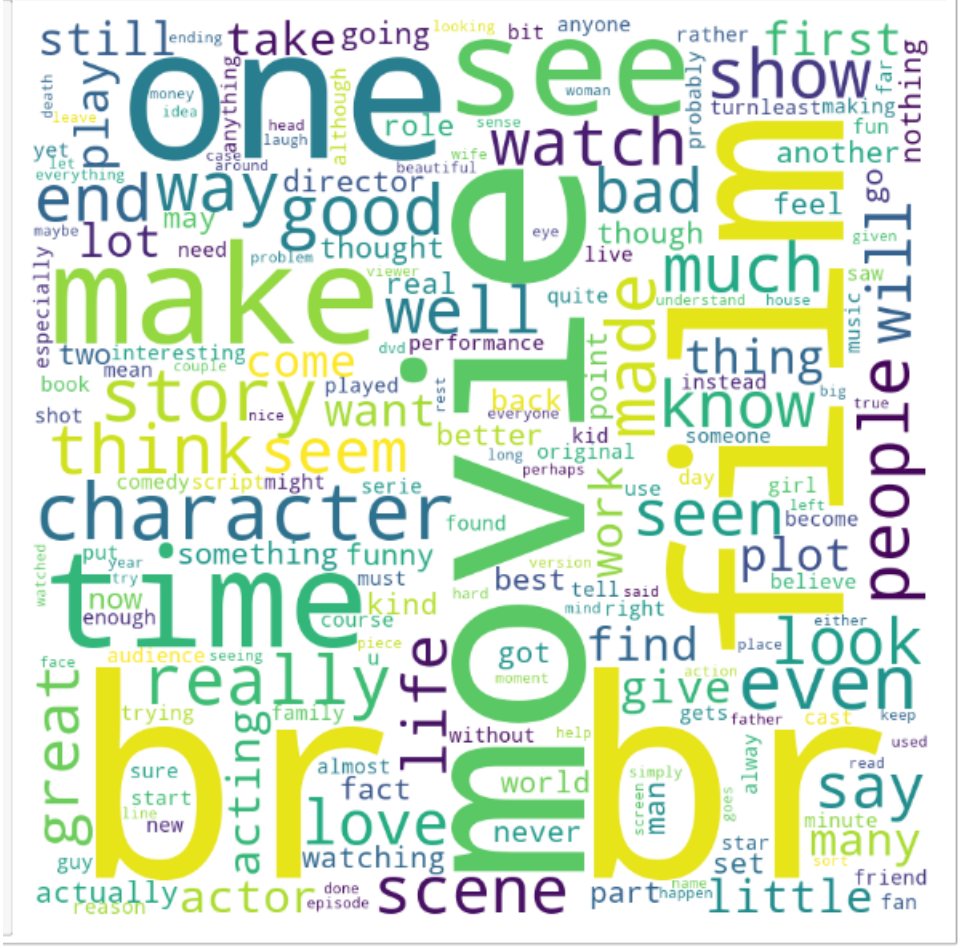
Step: 4 define the word cloud function

```
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)
```

Step 5: Plot the word cloud

```
# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



RESULT:

Thus, we done word cloud using word cloud package in python.

EX.NO:8	PYTHON KEYWORD EXTRACTION
DATE:11.09.23	

AIM:

To extract python key word.

INTRODUCTION TO KEY WORD EXTRACTION:

Keyword extraction in Python involves identifying and extracting the most important words or phrases from a text that represent its main topics or themes.

PROCEDURE:

Step 1: load the package for key word extraction

```
In [8]: from collections import Counter
        import re
```

Step 2: define the function

```
In [9]: def extract_keywords(text, num_keywords=5):
            # Tokenize the text into words
            words = re.findall(r'\b\w+\b', text.lower())

            # Calculate word frequencies
            word_frequencies = Counter(words)

            # Get the most common words as keywords
            keywords = [word for word, _ in word_frequencies.most_common(num_keywords)]

            return keywords
```

Step 3: Extract the key words

```
In [11]: text = '''This is a sample text for keyword extraction.
         This text is used to demonstrate keyword extraction in Python.'''
         keywords = extract_keywords(text, num_keywords=5)
         print("Keywords:", keywords)

         Keywords: ['this', 'is', 'text', 'keyword', 'extraction']
```

RESULT:

Thus, we extracted the key words using counter package.

EX.NO:9	NAMED ENTITY RECOGNITION
DATE:29.09.23	

AIM:

To do named entity recognition using spacy library.

INTRODUCTION TO NAMED ENTITY RECOGNITION:

Named entity recognition (NER) is a natural language processing (NLP) method that extracts information from text. NER involves detecting and categorizing important information in text known as names entities. Named entities refer to the key subjects of a piece of text, such as names, locations, companies, events and products, as well as themes, topics, times, monetary values and percentages.

PROCEDURE:

Step 1: Import the necessary packages

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

Step 2: take spacy input and do named entity recognition

Input:

```
sent=nlp('''In the 16th century, an age of great marine and terrestrial exploration, Ferdinand Magellan led the first
expedition to sail around the world. As a young Portuguese noble,he served the king of Portugal, but he became involved
in the quagmire of political intrigue at court and lost the king's favor. After he was dismissed from service by the king
of Portugal, he offered to serve the future Emperor Charles V of Spain.

A papal decree of 1493 had assigned all land in the New World west of 50 degrees W longitude to Spain and all the land east
of that line to Portugal. Magellan offered to prove that the East Indies fell under Spanish authority. On September 20, 1519,
Magellan set sail from Spain with five ships. More than a year later, one of these ships was exploring the topography of South
America in search of a water route across the continent.

This ship sank, but the remaining four ships searched along the southern peninsula of South America.
Finally they found the passage they sought near 50 degrees S latitude. Magellan named this passage the Strait of All Saints,
but today it is known as the Strait of Magellan.

One ship deserted while in this passage and returned to Spain, so fewer sailors were privileged to gaze at that first
panorama of the Pacific Ocean. Those who remained crossed the meridian now known as the International Date Line in the early
spring of 1521 after 98 days on the Pacific Ocean. During those long days at sea, many of Magellan's men died of starvation and
disease.

Later, Magellan became involved in an insular conflict in the Philippines and was killed in a tribal battle.
Only one ship and 17 sailors under the command of the Basque navigator Elcano survived to complete the westward journey to
Spain and thus prove once and for all that the world is round, with no precipice at the edge.''')
for ent in sent.ents:
    print(ent.text,ent.label_)
```

Output:

```
    print(ent.text,ent.label_)
```

```
the 16th century DATE
Ferdinand Magellan PERSON
first ORDINAL
Portuguese NORP
Portugal GPE
Portugal GPE
Charles V PERSON
Spain GPE
1493 DATE
50 degrees QUANTITY
Spain GPE
Portugal GPE
Magellan PRODUCT
the East Indies ORG
Spanish NORP
September 20, 1519 DATE
Magellan PRODUCT
Spain GPE
five CARDINAL
More than a year later DATE
one CARDINAL
South America LOC
four CARDINAL
South America LOC
50 degrees QUANTITY
Magellan PRODUCT
the Strait of All Saints LOC
today DATE
the Strait of Magellan LOC
One CARDINAL
Spain GPE
first ORDINAL
the Pacific Ocean LOC
the International Date Line EVENT
the early spring of 1521 DATE
98 days DATE
the Pacific Ocean LOC
those long days DATE
Magellan PRODUCT
Magellan PRODUCT
Philippines GPE
Only one CARDINAL
17 CARDINAL
Basque NORP
Elcano PERSON
Spain GPE
```

RESULT:

Thus, we done named entity recognition with spacy.

EX.NO:10	LATENT SEMANTIC ANALYSIS
DATE:03.10.23	

AIM:

To do LATENT semantic analysis.

INTRODUCTION TO LATENT ANALYSIS:

Latent Semantic Analysis is a natural language processing method that analyzes relationships between a set of documents and the terms contained within. It uses singular value decomposition, a mathematical technique, to scan unstructured data to find hidden relationships between terms and concepts.

LSA is primarily used for concept searching and automated document categorization. However, it's also found use in software engineering (to understand source code), publishing (text summarization), search engine optimization, and other applications.

There are a number of drawbacks to Latent Semantic Analysis, the major one being is its inability to capture polysemy (multiple meanings of a word). The vector representation, in this case, ends as an average of all the word's meanings in the corpus. That makes it challenging to compare documents.

PROCEDURE:

Step 1: import the necessary packages.

```
In [8]: from sklearn.feature_extraction.text import TfidfVectorizer
        from sklearn.decomposition import TruncatedSVD
        from sklearn.datasets import fetch_20newsgroups
```

Step 2: load the dataset

```
In [9]: newsgroups_data = fetch_20newsgroups(subset='all', remove=('headers', 'footers', 'quotes'))
        corpus = newsgroups_data.data
```

Step 3: Do vectorization

```
In [10]: # Step 1: TF-IDF Vectorization
         vectorizer = TfidfVectorizer(max_df=0.5, min_df=2, stop_words='english', use_idf=True)
         tfidf_matrix = vectorizer.fit_transform(corpus)
```

Step 4: Do latent sementic analysis

```
In [11]: # Step 2: Latent Semantic Analysis (LSA)
         num_topics = 10  # Specify the number of topics
         lsa = TruncatedSVD(n_components=num_topics, random_state=42)
         lsa_matrix = lsa.fit_transform(tfidf_matrix)
```

Step 5: display the most important terms for each topic

```
In [12]: # Display the most important terms for each topic
         terms = vectorizer.get_feature_names_out()
         for i, topic in enumerate(lsa.components_):
             top_terms_idx = topic.argsort()[-10:][::-1]
             top_terms = [terms[idx] for idx in top_terms_idx]
             print(f"Topic {i + 1}: {', '.join(top_terms)}\n")
```

```
Topic 1: don, just, like, know, people, think, does, use, time, good

Topic 2: windows, thanks, drive, card, dos, file, pc, software, scsi, program

Topic 3: god, windows, jesus, does, bible, thanks, christ, christian, dos, faith

Topic 4: drive, scsi, god, ide, card, controller, hard, drives, game, disk

Topic 5: drive, key, scsi, government, chip, encryption, clipper, people, keys, ide

Topic 6: windows, dos, file, think, problem, drive, os, window, run, disk

Topic 7: know, thanks, don, does, just, like, drive, car, people, advance

Topic 8: key, game, chip, does, god, clipper, encryption, keys, know, team

Topic 9: geb, edu, dsl, cadre, n3jxp, pitt, chastity, skepticism, intellect, shameful

Topic 10: car, just, bike, 00, good, ve, new, god, engine, like
```

RESULT:

Thus, we did the LATENT semantic analysis.

EX.NO:11	
DATE:10.10.23	**DETERMINE OPTIMUM NUMBER OF TOPICS IN A DOCUMENT**

AIM:

To determine optimum number of topics in a document.

INTRODUCTION:

To decide on a suitable number of topics, you can compare the goodness-of-fit of LDA models fit with varying numbers of topics. You can evaluate the goodness-of-fit of an LDA model by calculating the perplexity of a held-out set of documents. The perplexity indicates how well the model describes a set of documents. A lower perplexity suggests a better fit.

PROCEDURE:

Step 1: Import the necessary packages

In [1]:
```
!pip install gensim
```

```
Requirement already satisfied: gensim in c:\users\jeevi\anaconda3\lib\site-packages (4.3.0)
Requirement already satisfied: smart-open>=1.8.1 in c:\users\jeevi\anaconda3\lib\site-packages (from gensim) (5.2.1)
Requirement already satisfied: FuzzyTM>=0.4.0 in c:\users\jeevi\anaconda3\lib\site-packages (from gensim) (2.0.5)
Requirement already satisfied: numpy>=1.18.5 in c:\users\jeevi\anaconda3\lib\site-packages (from gensim) (1.23.5)
Requirement already satisfied: scipy>=1.7.0 in c:\users\jeevi\anaconda3\lib\site-packages (from gensim) (1.10.0)
Requirement already satisfied: pyfume in c:\users\jeevi\anaconda3\lib\site-packages (from FuzzyTM>=0.4.0->gensim) (0.2.25)
Requirement already satisfied: pandas in c:\users\jeevi\anaconda3\lib\site-packages (from FuzzyTM>=0.4.0->gensim) (1.5.3)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\jeevi\anaconda3\lib\site-packages (from pandas->FuzzyTM>=0.4.
0->gensim) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\jeevi\anaconda3\lib\site-packages (from pandas->FuzzyTM>=0.4.0->gensim)
(2022.7)
Requirement already satisfied: fst-pso in c:\users\jeevi\anaconda3\lib\site-packages (from pyfume->FuzzyTM>=0.4.0->gensim) (1.
8.1)
Requirement already satisfied: simpful in c:\users\jeevi\anaconda3\lib\site-packages (from pyfume->FuzzyTM>=0.4.0->gensim) (2.1
1.0)
Requirement already satisfied: six>=1.5 in c:\users\jeevi\anaconda3\lib\site-packages (from python-dateutil>=2.8.1->pandas->Fuz
zyTM>=0.4.0->gensim) (1.16.0)
Requirement already satisfied: miniful in c:\users\jeevi\anaconda3\lib\site-packages (from fst-pso->pyfume->FuzzyTM>=0.4.0->gen
sim) (0.0.6)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\jeevi\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\jeevi\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\jeevi\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\jeevi\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\jeevi\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\jeevi\anaconda3\lib\site-packages)
```

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from gensim.models import LdaModel
from gensim.corpora import Dictionary
from gensim.models.coherencemodel import CoherenceModel
from nltk.corpus import stopwords
import string
```

Step 2: Preprocess the data

In [2]:
```
# Sample text data
documents = [
    "Latent Semantic Analysis is a technique in natural language processing.",
    "It analyzes the relationships between words and documents.",
    "It helps uncover the hidden structure in the relationships between words."
]
```

In [3]:
```
# Preprocess the text data (tokenization, stopword removal, etc.)
stop_words = set(stopwords.words('english'))
translator = str.maketrans('', '', string.punctuation)

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

In [4]:
```
# Tokenize and preprocess the documents
processed_docs = [preprocess(doc) for doc in documents]
```

Step 3: Create the dictionary and corpus

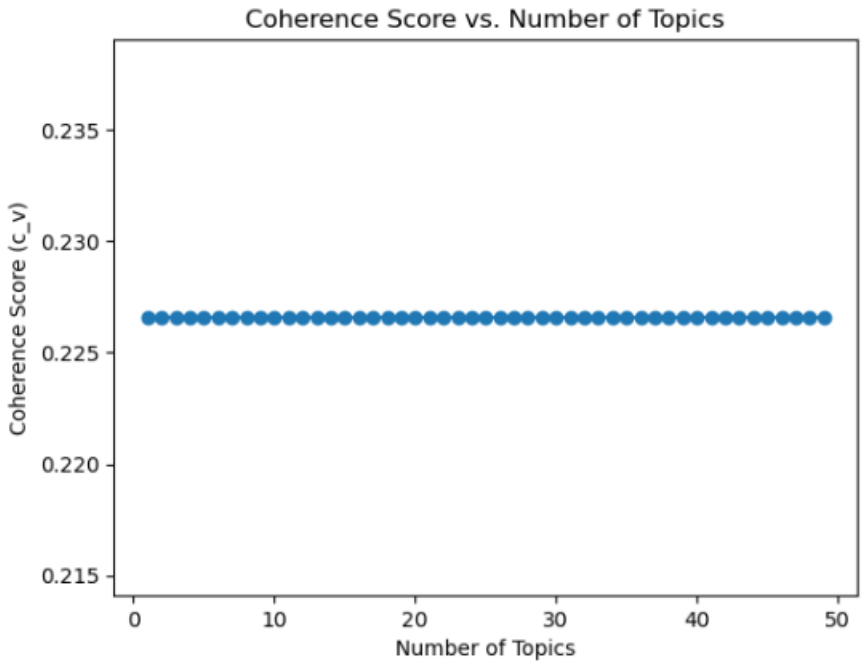
In [5]:
```
# Create a Gensim dictionary and corpus
dictionary = Dictionary(processed_docs)
corpus = [dictionary.doc2bow(doc) for doc in processed_docs]
```

In [6]:
```
# Function to calculate coherence for a given number of topics
def calculate_coherence(dictionary, corpus, texts, num_topics):
    lda_model = LdaModel(corpus=corpus, id2word=dictionary, num_topics=num_topics)
    coherence_model = CoherenceModel(model=lda_model, texts=texts, dictionary=dictionary, coherence='c_v')
    return coherence_model.get_coherence()
```

In [7]:
```
# Range of topics to try
min_topics = 1
max_topics = 50
step = 1
topics_range = range(min_topics, max_topics, step)
```

In [8]:
```
# Calculate coherence scores for different numbers of topics
coherence_scores = []
for num_topics in topics_range:
    coherence_score = calculate_coherence(dictionary, corpus, processed_docs, num_topics)
    coherence_scores.append(coherence_score)
```

In [9]:
```
# Plot the coherence scores
plt.plot(topics_range, coherence_scores, marker='o')
plt.xlabel("Number of Topics")
plt.ylabel("Coherence Score (c_v)")
plt.title("Coherence Score vs. Number of Topics")
plt.show()
```



In [10]:
```
# Find the optimum number of topics with the highest coherence score
optimal_num_topics = topics_range[np.argmax(coherence_scores)]
print("Optimal number of topics:", optimal_num_topics)
```

```
Optimal number of topics: 12
```

RESULT:

Thus, we find the optimum number of topics in a document.

EX.NO:12	FUNDAMENTALS OF TOPIC MODELING
DATE:17.10.23	

AIM:

To know the fundamentals of topic modeling.

INTRODUCTION:

Topic modeling is a part of NLP that is used to determine the topic of a set of documents based on the content and generate meaningful insights from the similar words in the entire corpus of text data, thereby performing documents-based contextual analysis to analyze the context.

An ongoing and pervasive issue in Data Science is the ability to automatically extract value from various sources without (or with little) a priori knowledge. Unsupervised machine learning techniques, like topic modeling, require less user input than supervised algorithms for text classification. This is because they don't require human training with manually tagged data.

PROCEDURE:

Topic modeling is a natural language processing (NLP) technique that is used to discover the hidden thematic structure in a collection of documents. It is particularly useful for tasks such as document clustering, summarization, and information retrieval. One of the most popular topic modeling algorithms is Latent Dirichlet Allocation (LDA), but there are others like Non-Negative Matrix Factorization (NMF) and Latent Semantic Analysis (LSA).

LDA (Latent Dirichlet Allocation):

Step 1: load the necessry packages.

In [11]:
```
# Load the required libraries and associated functions
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
```

Step 2: load the dataset

In [12]:
```
# Fetch dataset
dataset = fetch_20newsgroups(
    shuffle=True,
    random_state=1,
    remove=('headers', 'footers', 'quotes')
)
documents = dataset.data
```

Step 3: choose the number of features and number of topics

```
In [13]: # Initialise number of topics and features for experiment
         no_features = 1000
         no_topics = 10 # We choose a sample 10
```

Step 4: Preprocess the data

```
In [15]: # LDA can only use raw term counts for LDA because it is a probabilistic graphical model
         tf_vectorizer = CountVectorizer(
             max_df=0.95,
             min_df=2,
             max_features=no_features,
             stop_words='english')
         tf = tf_vectorizer.fit_transform(documents)
         tf_feature_names = tf_vectorizer.get_feature_names_out()
```

Step 5: Build the LDA model

```
In [16]: # Run LDA
         lda = LatentDirichletAllocation(
             n_components=no_topics,
             max_iter=5,
             learning_method='online',
             learning_offset=50.,
             random_state=0)
         lda.fit(tf)

         no_top_words = 5 # How many top words to display
```

Step 6: Display top words for each topic

```
In [17]: # Display top words for each topic
         for topic_idx, topic in enumerate(lda.components_):
             print("Topic %d:" % (topic_idx))
             print(" ".join([tf_feature_names[i] for i in topic.argsort()[:-no_top_words - 1:-1]]))
```

```
Topic 0:
people gun armenian armenians war
Topic 1:
government people law mr use
Topic 2:
space program output entry data
Topic 3:
key car chip used keys
Topic 4:
edu file com available mail
Topic 5:
god people does jesus say
Topic 6:
windows use drive thanks does
Topic 7:
ax max b8f g9v a86
Topic 8:
just don like think know
Topic 9:
10 00 25 15 12
```

RESULT:

Thus, we known the fundamental of topic modelling and LDA.