NLP models

cmd :- pip install nltk

IDLE :-

```
import nltk
nltk.download()
```

1. Tokenization

```
from nltk.tokenize import word_tokenize, sent_tokenize
string = """hi! how are you."""
sent_tokenize(string)
['hi!', 'how are you.']
word_tokenize(string)
['hi', '!', 'how', 'are', 'you', '.']
```

2. Stopwords

```
nltk.download("stopwords")
from nltk.corpus import stopwords
string = "ramu ate the apple!!"
strings = word_tokenize(string)
strings
['ramu', 'ate', 'the', 'apple', '!', '!']
stop_words = set(stopwords.words("english"))
filtered_list = []
for word in strings:
    if word.casefold() not in stop_words:
```

```
    filtered_list.append(word)
filtered_list
['rama', 'ate', 'apple', '!', '!']
```

4. Stemming

```
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
string = """ Sai Vaishnavi is talking """
words = word_tokenize(string)
words
['sai', 'vaishnavi', 'is', 'talking']
stemmed_words = [stemmer.stem(word) for word in words]
stemmed_words
['sai', 'vaishnavi', 'is', 'talk']
```

5. Tagging parts of speech

```
from nltk.tokenize import word_tokenize
string = """ If you wish to make an apple pie from scratch,
    you must first invent the universe."""
words = word_tokenize(string)
nltk.pos_tag(words)
[('If', 'IN'),
 ('you', 'PRP'),
 ('wish', 'VBP'),
 ('to', 'TO'),
 ('make', 'VBP'),
```

```
('an', 'DT'),
('apple', 'NN'),
('pie', 'NN'),
('from', 'IN'),
('scratch', 'NN'),
(',', ','),
('you', 'PRP')
('must', 'MD')
('first', 'VB')
('invent', 'VB')
('the', 'DT')
('universe', 'NN')
('.', '.')]
```

6. Lemmatizing

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
lemmatizer.lemmatize('scraves')
'scarf'
sting = "Bhavana loves apples"
words = word_tokenize(sting)
words
['Bhavana', 'loves', 'apples']
lemmatized_words = [lemmatizer.lemmatize(word) for word
in words]
lemmatized_words
['Bhavana' 'love' 'apple']
```

Spell checking

1. Text Blob (standard spell checker) [pip install TextBlob]

```
import textblob
from textblob import TextBlob
tb = TextBlob("I want football")
tb.conect()
TextBlob("I want football")
tb = TextBlob("I want to play footbal")
tb.correct()
TextBlob("I want to pale football")
tb = TextBlob("python is eays to learn")
tb.conect()
TextBlob("Platon is days to learn")
```

2. Custom spell checker (spello) [pip install spello]

```
from spello.model import SpellCorrectionModel
sp = SpellConectionModel(language="en")
with open("c:\users\sample.txt","r") as file:
    data = file.readlines()
    data = [i.strip() for i in data]
    data
sp.train(data)
```

```
Spello training started...
Context model training started...
Symspell training started...
phoneme training started...
```

Spello training completed successfully...

```
sp = spell_correct('spell chekcing in nltk")
{'original_text':'spell chekcing in nltk', 'spell_corrected_
text':'spell checking in nltk', 'correction_dict': {'spell':
'spell', 'checking': 'checking', 'nltk': 'nltk'}.
```

Sentence Segmentation

```
import spacy   Pip install spacy :-cmd
nlp = spacy.load("en_core_web_lg")
nlp
<spacy.lang.en.English object at 0x00000020E99393710>
text = " This is 1st sentence, this is 2nd sentence."
doc = nlp(text)
doc = nlp(text)
doc
'This is 1st sentence, this is 2nd sentence.'
for i in doc.sents:
    print(i.text)
This is 1st sentence.
this is 2nd sentence.
```

Word Frequency distribution

```
import nltk
from nltk.probability import FreqDist
text = """I am harshitha. I love cricket."""
words = word_tokenize(text)
len(words)
 8
len(FreqDist(words))
 6
FreqDist(words)
<FreqDist with 6 samples and 8 outcomes>
FreqDist(words).most_common(3)
[('I',2),('.',2),('am',1)]
FreqDist(words).freq("I")
2
```

Word Segmentation

```
pip install matplotlib
import matplotlib.pyplot as plt
fig,ax = plt.subplots(figsize=(10,5))
FreqDist(words).plot(15)
<Axes Subplot: xlabel="samples", ylabel="counts">
print(FreqDist(words))
[('harshitha',1),('am',2),('I',2),('love',1),('cricket',1)]
```

Regd. No. (AUTONOMOUS) R.V.R. & J.C. College of Engineering Page No. 8

Count Vectorizer

```
import sklearn
from sklearn.feature_extraction.text import CountVectorizer
text = ["this is first document, this is second sentence."]
text
['this is first sentence', 'this is second sentence.']
vectorizer = countVectorizer()
vectorizer.corp fit(text)
CountVectorizer()
x = vectorizer.fit_transform(text)
x
<3x6 sparse matrix of type '<class' numpy.int64'>' with
12 stored elements in compressed Sparse Row format>
vectorizer.vocabulary_
{'this': 4, 'is': 2, 'first': 1, 'document': 0, 'second': 3}
vectorizer.get_feature_names()
['document', 'first', 'is', 'second', 'this']
x.toarray()
array([[1, 1, 1, 0, 1],
       [1, 0, 1, 1, 1]], dtype=int64)
vectorizer.vocabulary_get("this")
4
pip install pandas
import pandas as pd
```

```
matrix = pd.DataFrame(x.toarray(), columns = vectorizer.get
-feature_names())
matrix
```

	document	first	is	second	this
0	1	1	1	0	1
1	1	0	1	1	1
2					

White SpaceTokenizer

```
import nltk
from nltk.tokenize import WhitespaceTokenizer
wst = WhitespaceTokenizer()
text = "this is is in first is document in from nltk is."
whitespace = wst.tokenize(text)
whitespace
```

```
['this', 'is', 'first', 'document', 'from', 'nltk', '.']
```

N-Gram model

```
from sklearn.feature_extraction.text import CountVectorizer
text=["this is my new sentence."]
n_gram = CountVectorizer(ngram_range=(2,2))
trans = n_gram.fit_transform(text)
trans
<1x4 sparse matrix of type '<class' numpy.int64'>' with
 4 stored elements in Compressed Sparse Row format>
n_gram.vocabulary_
{'this is': 3, 'is my': 0, 'my new': 1, 'new sentence':2
trans.toarray()
array([[1,1,1,1]], dtype=int64)
n_gram=CountVectorizer(ngram_range=(1,3))
trans=n_gram.fit_transform(text)
n_gram.vocabulary_
{'this': 9, 'is':0, 'my':3, 'new':6, 'sentence':8, 'this is
:10, 'is my':1, 'my new':4, 'new sentence':7, 'this is
my': 11, 'is my new': 2, 'my new sentence':5}
trans.toarray()
array([[1,1,1,1,1,1,1,1,1,1,1,1]], dtype=int64)
n_gram.vocabulary_.get("this is")
10
```

Bag of words

trans shape

(1,12)

```
t = ["I am learning nlp"]
t
['I am learning nlp']
n_gram.transform(t).toarray()
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]], dtype=int64)
t = ["i am learning my nlp"]
n_gram.transform(t).toarray()
array([[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]], dtype=int64)
trans = n_gram.fit_transform(text+t)
n_gram.vocabulary_
{'this': 17, 'is': 3, 'my': 9, 'new': 13, 'sentence': 16, 'this
is': 18, 'is my': 4, 'my new': 10, 'new sentence': 14, 'this
is my': 19, 'is my new': 5, 'my new sentence': 11, 'am':
0, 'learning': 6, 'nlp': 15, 'am learning': 1, 'learning my':
7, 'my nlp': 12, 'am learning my': 2, 'learning my nlp':
n_gram.transform(text+t).toarray()
array([[0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1],
       [1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0]],
      dtype=int64)
```

Inverse transformations

pip install sklearn :- cmd

```
import sklearn
from sklearn.feature_extraction.text import CountVectorizer
text=["I am alaparthi harshitha"]
vectorizer=CountVectorizer()
x=vectorizer.fit_transform(text)
x
<1x3 sparse matrix of type '<class 'numpy.int64'>' with 3
stored elements in compressed row format>
vectorizer.vocabulary_
{'am':1,'alaparthi':0,'harshitha':2}
y=vectorizer.inverse_transform(x)
y
[array(['am','alaparthi','harshitha'],dtype='<U9')]
```

Tom frequency and Inverse Document Frequency (TF and IDF)

```
pip install sklearn :- cmd
import sklearn
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd
corpus = ["my name is harshitha", "I am studying btech",
"this is my data"]
corpus
['my name is harshitha', 'I am studying btech', 'this is my
data']
corpus[1]
I am studying btech
tfidf = TfidfVectorizer()
x = tfidf.fit_transform(corpus)
x
<3x9 sparse matrix of type '<class' numpy.float64'>'
with 11 stored elements in compressed Row format>
tfidf.vocabulary_
{'my': 5, 'name': 6, 'is': 4, 'harshitha': 3, 'am': 0, 'studyi-
ng': 7, 'btech': 2, 'this': 8, 'data': 1}
tfidf.get_feature_names()
['am', 'data', 'btech', 'harshitha', 'is', 'my', 'name', 'studying']
```

```
tfidf.idf_

array([1.69314 , 1.693141 , 1.693147, 1.69314, 1.287,
       1.2876, 1.6931, 1.6931, 1.69317])

matrix = pd.DataFrame(x.toarray(), columns = tfidf.get_
feature_names())

matrix

   am  data  btech  ...  name  studying  this
0 0.0  0.00   0.00  ...  0.56      0.00  0.00
1 0.57 0.00   0.57  ...  0.00      0.57  0.00
2 0.00 0.56   0.00  ...  0.00      0.00  0.56

[3 rows x 9 columns]
```

POS tagging

Pip install nltk :- cmd

import nltk

nltk.help.upenn_tagset("MD")

MD: modal auxiliary

text = "I am harshitha."

words = word_tokenize(text)

words

['I', 'am', 'harshitha', '.']

len(words)

4

nltk.pos_tag(words)

[('I', 'PRP'), ('am', 'VBP'), ('harshitha', 'RB'), ('.', '.')]

nltk.help.brown_tagset()

(: opening parenthesis

(

): closing parenthesis

)

*: regular

not n't

.

.

.

Multiword Expressions

```
import nltk
nltk.download('punkt')
true
from nltk.tokenize import MWETokenizer
from nltk.tokenize import word_tokenize
mwe = MWETokenizer([('new', 'york'), ('hong', 'kong'), ('takes
      ('off')], separator="_")
text1 = "the flight is going from new york to hongkong."
text2 = "the plane takess off from new york at 10am."
mwe1 = mwe.tokenize(word_tokenize(text1))
mwe2 = mwe.tokenize(word_tokenize(text2))
mwe1
['The', 'flight', 'is', 'going', 'from', 'new_york', 'to', 'hong-
 kong']
mwe2
['The', 'plane', 'takes_off', 'from', 'new_york', 'at', '10',
 'am']
```

cleaning text data

pip install clean_text : cmd

import cleantext

from cleantext import clean

Unicode

s1 = "2ürick"

clean(s1, fix_unicode = True)

'zurick'

Asciicode

s2 = "kO\u017eu\u015fad\u0131k"

clean(s2, to_ascii = True)

'kozusadik'

Lowercase

s3 = "My nAme is HarShitHa"

clean(s3, lower = True)

'my name is harshitha'

URL replacement

s4 = "https://www.Google.com has surpassed https://www.Bing.com in search volume."

clean(s4, no_urls = True, replace_with_url = "URL")

'URL has surpassed URL in search volume'

Replace currency

```
s5 = "I want $ 40"
clean(s5, no_currency_symbols=True, replace_with_currency_symbol="Rupees")
'I want Rupees 40'
```

No punctuations

```
s6 = "40,000 is more than 30,000"
clean(s6, no_punct=True)
'40000 is more than 30000'
s7 = "40,000 is more than 30,000"
clean(s7, no_punct=True, replace_with_punct="@")
'40@000 is more than 30@000
```

Digit Replacement

```
s8 = "rtegrtrty236 78jyhjhi"
clean(s8, no_digits=True, replace_with_digit=" ")
'rtegrtrty jyhjhi'
```

1. Chuncking

```
import nltk
from nltk.tokenize import word_tokenize
string = "It's a dangerous business, Frodo, going out your
        door."
words = word_tokenize(string)
pos = nltk.pos_tag(words)
grammar = "NP: {<DT>?<JJ>*<NN>}"
chunk_parser = nltk.RegexpParser(grammar)
tree = chunk_parser.parse(pos))
tree.draw()
```

2. Chinking

```
chunk_grammar = """ chunk : {<.*>+} }<JJ>{"""
chunk_parser = nltk.RegexpParser(chunk_grammar)
tree = chunk_parser.parse(pos)
tree.draw()
res = chunk_parser.parse(pos)
print(res)
```

```
(S
   it/PRP
   is/VBZ
   a/DT
   :
   door/NN)
```

Matplot Library

```
pip install matplotlib :- cmd
import matplotlib.pyplot as plt
x = [10, 20, 30, 40]
y = [20, 25, 35, 55]
plt.plot(x, y)
[<matplotlib.lines.Line2D object at 0x00016>]
plt.show()
```

Adding title

```
x = [10, 20, 30, 40]
y = [20, 25, 35, 55]
plt.plot(x, y)
[<matplotlib.lines.Line2D object at 0x00>]
plt.title("Linear Graph")
Text(0.5, 1.0, 'Liner Graph')
plt.show()
```

Adding color

```
x = [10, 20, 30, 40]
y = [20, 30, 40, 50]
plt.plot(x, y)
plt.title("Linear graph", fontsize = 25, "color" = "green")
plt.show()
```

Adding labels

```
x = [10, 20, 30, 40]
y = [20, 30, 40, 50]
plt.plot(x, y)
plt.title("Linear graph", fontsize = 25, color = "green")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show()
```

ylim and xticks

```
x = [10, 20, 30, 40]
y = [20, 30, 40, 50]
plt.ylim(0, 60)
plt.xticks(x, labels = ["one", "two", "three", "four"])
plt.plot(x, y)
plt.show()
```

Legend

```
x = [10, 20, 30, 40]
y = [20, 34, 65, 34]
plt.plot(x, y)
plt.title("Linear graph")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.legend(["GFG"])
```

Adding labels

```
x = [10, 20, 30, 40]
y = [20, 30, 40, 50]
plt.plot(x, y)
plt.title("Linear graph", fontsize = 25, color = "green")
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.show()
```

ylim and xticks

```
x = [10, 20, 30, 40]
y = [20, 30, 40, 50]
plt.ylim(0, 60)
plt.xticks(x, labels = ["one", "two", "three", "four"])
plt.plot(x, y)
plt.show()
```

Legend

```
x = [10, 20, 30, 40]
y = [20, 34, 65, 34]
plt.plot(x, y)
plt.title("Linear graph")
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.legend(["GFG"])
plt.show()
```

Figure class

```
import matplotlib.pyplot as plt
from matplotlib.figure import figure
x = [10, 20, 30, 40]
y = [20, 25, 35, 55]
fig = plt.figure(figsize=(7,5), facecolor='g', edgecolor='b',
linewidth=7)
ax = fig.add_axes([1,1,1,1])
ax.plot(x,y)
```

Multiple plots

```
x = [10, 20, 30, 40]
y = [10, 20, 40, 30]
fig = plt.figure(figsize=(5,4))
ax = fig.add_axes([1,1,1,1])
ax1 = ax.plot(x,y)
ax2 = ax.plot(y,x)
ax.set_title("Multiple Graphs")
ax.set_xlabel("X_axis")
ax.set_ylabel("Y_axis")
ax.legend(labels = ['line1', 'line2'])
plt.show()
```

```
→ import matplotlib.pyplot as plt
  from matplotlib.figure import figure
  x=[10,20,30,40]
  y=[20,25,35,55]
  height = 4inches
  fig = plt.figure(figsize=(5,4))
  ax1=fig.add_axes([0.1,0.1,0.8,0.8])
  ax2 = fig.add_axes([1,0.1,0.8,0.8])
  ax1.plot(x,y)
  ax2.plot(y,x)
  plt.show()
```

Linechart

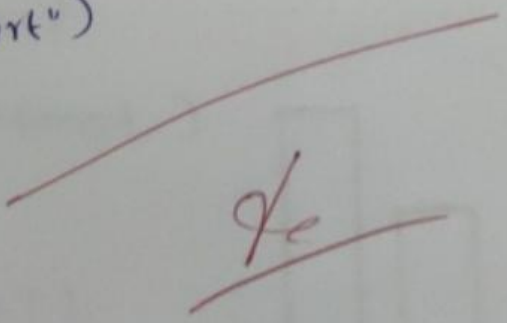
```
import matplotlib.pyplot as plt
x=[10,20,30,40]
y=[20,25,35,55]
plt.plot(x,y, color="green", linewidth=3, marker='o', marker
size=15, linestyle='--')
plt.title("Linechart")
plt.xlabel("x_axis")
plt.ylabel("y_axis")
plt.show()
```

Barchart

```
import matplotlib.pyplot as plt
x = [10, 20, 30, 40]
y = [20, 30, 40, 50]
plt.bar(x, y, color="green", width=3)
plt.title("Bar Graph")
plt.show()
```

→

```
import matplotlib.pyplot as plt
x = [10, 20, 30, 40]
y = [20, 30, 40, 50]
plt.bar(x, y, color="green", width=3, edgecolor="blue", linewidth=2)
plt.title("Barchart")
plt.show()
```



hmmlearn

```
pip install hmmlearn
import hmmlearn
from hmmlearn.hmm import CategoricalHMM
import numpy as np
startprob = np.array([0.5, 0.5])
transmat = np.array([[0.7, 0.3], [0.3, 0.7]])
covar = np.array([[0.9, 0.1], [0.2, 0.8]])
model = CategoricalHMM(n_components=2, startprob_prior=
startprob, transmat_prior = transmat)
x = [[0, 1, 0, 1], [0, 0, 0, 0], [1, 1, 1, 1], [1, 0, 0, 0]]
model.fit(x)
CategoricalHMM( )
print(model.transmat_)
[[0.   1.   ]
 [0.800 0.19952]]
prob = model.decode(np.array([0, 1, 0, 1]).reshape(4, 1),
lengths = None, algorithm = None)
print(prob)
(-2.00511933, array([1, 0, 1, 0], dtype=int64))
print(np.exp(prob[0]))
0.097656
```

```
x, z = model.sample(10)

print(x)

tensor([[0.],
        [0.],
        [0.],
        [0.],
        [1.],
        [0.],
        [1.],
        [0.],
        [0.],
        [1.]])

print(z)

tensor([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

Word Embeeding in NLP (Word2Vec)

cmd : pip install gensim

```
import gensim
from gensim.models import Word2Vec
import warnings
warnings.filterwarnings('ignore')
st = [['this', 'is', 'sample1'], ['this', 'is', 'sample2'],
      ['this', 'is', 'sample3']]
print(st)
```

```
[['this', 'is', 'sample1'],
 ['this', 'is', 'sample2'],
 ['this', 'is', 'sample3']]
```

```
model = Word2Vec(st, min_count=1)
print(model)
```

```
Word2Vec(vocab=5, vector_size=100, alpha=0.025)
```

```
words = list(model.wv.key_to_index)
print(words)
```

```
['is', 'this', 'sample3', 'sample2', 'sample1']
```