Natural Language Processing

NLP Pipeline :

GFG : https://www.geeksforgeeks.org/natural-language-processing-nlp-pipeline/

1. Data Acquisition
2. Text Cleaning
3. Text Preprocessing
4. Feature Engineering
5. Model Building
6. Evaluation
7. Deployment

1. Data Acquisition

Data acquisition is the process of collecting and recording data from various sources.

2.Text Cleaning

Github:
https://github.com/hrithikM86/Natural_Language_Processing/blob/main/1.Text%20Preprocessing.ipynb

Sometimes our acquired data is not very clean. it may contain HTML tags, spelling mistakes, or special characters

Lowercasing

```
df['column'][3].lower()
df['column'] = df['column'].str.lower()
```

Removing HTML tags

The given code is a Python function that uses regular expressions to remove HTML tags from a given text. The function `remove_html_tags` takes a `text` parameter and returns the text with all HTML tags removed.

```
import re
#re=Regular_expressions
```

```python
def remove_html_tags(text):
    pattern = re.compile('<.*?>')
    return pattern.sub(r'', text)
```

Removing URLs

The provided code is a Python function that utilises regular expressions to remove URLs from a given text. The function `remove_url` takes a `text` parameter and returns the text with all URLs removed. The regular expression pattern `r'https?://\S+|www\.\S+'` is used to identify and remove URLs from the text.

```python
def remove_url(text):
    pattern = re.compile(r'https?://\S+|www\.\S+')
    return pattern.sub(r'', text)
```

Removing Punctuations

```python
import string
string.punctuation
# sting.punctuations contains this =
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'

exclude = string.punctuation
def remove_punc(text):
    for char in exclude:
        text = text.replace(char,'')
    return text
```

This function may take time to run, an another alternative would be:

```python
def remove_punc1(text):
    return text.translate(str.maketrans('', '', exclude))
```

Chat words Treatment

```
chat_words = {"IMHO":'In my humble opinion',"FYI":'For your information'}

def chat_conversion(text):
    new_text = []
    for w in text.split():
        if w.upper() in chat_words:
            new_text.append(chat_words[w.upper()])
        else:
            new_text.append(w)
    return " ".join(new_text)
```

Spelling Corrections

```
from textblob import TextBlob

incorrect_text = 'ceertain conditionas duriing seveal ggenerations aree moodified in the saame maner.'

textBlb = TextBlob(incorrect_text)

textBlb.correct().string
```

Removing Stopwords

```
from nltk.corpus import stopwords
stopwords.words('english')
def remove_stopwords(text):
    new_text = []

    for word in text.split():
        if word in stopwords.words('english'):
```

```
                new_text.append('')
            else:
                new_text.append(word)
        x = new_text[:]
        new_text.clear()
        return " ".join(x)
```

Removing Emoji

```
import re
def remove_emoji(text):
    emoji_pattern = re.compile("["
                               u"\U0001F600-\U0001F64F"
                           u"\U0001F300-\U0001F5FF"
                           u"\U0001F680-\U0001F6FF"
                           u"\U0001F1E0-\U0001F1FF"
                           u"\U00002702-\U000027B0"
                           u"\U000024C2-\U0001F251"
                           "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'', text)
```

Alternative

```
import emoji
print(emoji.demojize('Python is 🔥'))
```

3. Text Preprocessing

Github :
https://github.com/hrithikM86/Natural_Language_Processing/blob/main/1.Text%20Preprocessing.ipynb

Tokenization

Tokenization is the process of segmenting the text into a list of tokens. In the case of sentence tokenization, the token will be sentenced and in the case of word tokenization, it will be the word. It is a good idea to first complete sentence tokenization and then word tokenization, here output will be the list of lists. Tokenization is performed in each & every NLP pipeline.

1. Using the split function

In [57]:
```python
# word tokenization
sent1 = 'I am going to delhi'
sent1.split()
```

Out[57]:
```
['I', 'am', 'going', 'to', 'delhi']
```

In [58]:
```python
# sentence tokenization
sent2 = 'I am going to delhi. I will stay there for 3 days. Let\'s hope the trip to be great'
sent2.split('.')
```

Out[58]:
```
['I am going to delhi',
 ' I will stay there for 3 days',
 " Let's hope the trip to be great"]
```

In [59]:
```python
# Problems with split function
sent3 = 'I am going to delhi!'
sent3.split()
```

Out[59]:
```
['I', 'am', 'going', 'to', 'delhi!']
```

In [60]:
```python
sent4 = 'Where do think I should go? I have 3 day holiday'
sent4.split('.')
```

Out[60]:
```
['Where do think I should go? I have 3 day holiday']
```

2. Regular Expression

In [61]:
```python
import re
sent3 = 'I am going to delhi!'
tokens = re.findall("[\w']+", sent3)
tokens
```

Out[61]: ['I', 'am', 'going', 'to', 'delhi']

In [62]:
```python
text = """Lorem Ipsum is simply dummy text of the printing and typesetting industry?
Lorem Ipsum has been the industry's standard dummy text ever since the 1500s,
when an unknown printer took a galley of type and scrambled it to make a type specimen book."""
sentences = re.compile('[.!?] ').split(text)
sentences
```

Out[62]:
```
['Lorem Ipsum is simply dummy text of the printing and typesetting industry',
 "\nLorem Ipsum has been the industry's standard dummy text ever since the 1500s, \nwhen an unknown printer took a galley of type and scrambled it to make a type specimen book."]
```

3. NLTK

In [63]:
```python
from nltk.tokenize import word_tokenize,sent_tokenize
```

In [64]:
```python
sent1 = 'I am going to visit delhi!'
word_tokenize(sent1)
```

Out[64]: ['I', 'am', 'going', 'to', 'visit', 'delhi', '!']

In [65]:
```python
text = """Lorem Ipsum is simply dummy text of the printing and typesetting industry?
Lorem Ipsum has been the industry's standard dummy text ever since the 1500s,
when an unknown printer took a galley of type and scrambled it to make a type specimen book."""

sent_tokenize(text)
```

Out[65]:
```
['Lorem Ipsum is simply dummy text of the printing and typesetting industry?',
 "Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, \nwhen an unknown printer took a galley of t
ype and scrambled it to make a type specimen book."]
```

In [66]:
```python
sent5 = 'I have a Ph.D in A.I'
sent6 = "We're here to help! mail us at nks@gmail.com"
sent7 = 'A 5km ride cost $10.50'

word_tokenize(sent5)
```

Out[66]: ['I', 'have', 'a', 'Ph.D', 'in', 'A.I']

4. Spacy

```
In [69]: import spacy
         nlp = spacy.load('en_core_web_sm')     #This is a dictionary
```

```
In [70]: doc1 = nlp(sent5)
         doc2 = nlp(sent6)
         doc3 = nlp(sent7)
         doc4 = nlp(sent1)
```

```
In [71]: for token in doc4:
             print(token)

I
am
going
to
visit
delhi
!
```

Stemming

Stemming and lemmatization are used to reduce words to their base form, which can help reduce the vocabulary size and simplify the text. Stemming involves stripping the suffixes from words to get their stem, whereas lemmatization involves reducing words to their base form based on their part of speech. This step is commonly used in various NLP tasks such as text classification, information retrieval, and topic modelling

```
In [72]: from nltk.stem.porter import PorterStemmer
```

```
In [73]: ps = PorterStemmer()
         def stem_words(text):
             return " ".join([ps.stem(word) for word in text.split()])
```

```
In [74]: sample = "walk walks walking walked"
         stem_words(sample)

Out[74]: 'walk walk walk walk'
```

```
In [75]: text = 'probably my alltime favorite movie a story of selflessness sacrifice and dedication to a noble cause but its not pre
         print(text)

probably my alltime favorite movie a story of selflessness sacrifice and dedication to a noble cause but its not preachy or bo
ring it just never gets old despite my having seen it some 15 or more times in the last 25 years paul lukas performance brings
tears to my eyes and bette davis in one of her very few truly sympathetic roles is a delight the kids are as grandma says more
like dressedup midgets than children but that only makes them more fun to watch and the mothers slow awakening to whats happen
ing in the world and under her own roof is believable and startling if i had a dozen thumbs theyd all be up for this movie
```

```
In [76]: stem_words(text)

Out[76]: 'probabl my alltim favorit movi a stori of selfless sacrific and dedic to a nobl caus but it not preachi or bore it just nev
er get old despit my have seen it some 15 or more time in the last 25 year paul luka perform bring tear to my eye and bett d
avi in one of her veri few truli sympathet role is a delight the kid are as grandma say more like dressedup midget than chil
dren but that onli make them more fun to watch and the mother slow awaken to what happen in the world and under her own roof
is believ and startl if i had a dozen thumb theyd all be up for thi movi'
```

Lemmatization

Like stemming but the returned word is always a english word unlike stemming

```python
import nltk
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()

sentence = "He was running and eating at same time. He has bad habit of swimming after playing long hours in the Sun."
punctuations="?:!.,;"
sentence_words = nltk.word_tokenize(sentence)
for word in sentence_words:
    if word in punctuations:
        sentence_words.remove(word)

sentence_words
print("{0:20}{1:20}".format("Word","Lemma"))
for word in sentence_words:
    print ("{0:20}{1:20}".format(word,wordnet_lemmatizer.lemmatize(word,pos='v')))
```

```
Word                Lemma
He                  He
was                 be
running             run
and                 and
eating              eat
at                  at
same                same
time                time
He                  He
has                 have
bad                 bad
habit               habit
of                  of
swimming            swim
after               after
playing             play
long                long
hours               hours
in                  in
the                 the
Sun                 Sun
```

POS tagging

Github :
https://github.com/hrithikM86/Natural_Language_Processing/blob/main/5.%20pos-tagging.ipynb

POS tagging involves assigning a part of speech tag to each word in a text. This step is commonly used in various NLP tasks such as named entity recognition, sentiment analysis, and machine translation

The model used for POS tagging are:

Youtube :
https://www.youtube.com/watch?v=269IGagoJfs&list=PLKnIA16_RmvZo7fp5kkIth6nRTeQQsjfX&index=7

- Hidden Markov model -> used for Part of Speech Tagging
- Emission Probability,Transition Probability
- Viterbi Algorithm -> Used to improve the model

```
In [49]: import spacy

In [50]: nlp = spacy.load('en_core_web_sm')                    #Loading the dictionary

In [11]: doc = nlp(u"I will google about facebook")            #This will do the POS tagging

In [51]: doc.text

Out[51]: 'I will google about facebook'

In [54]: doc[-1]

Out[54]: facebook

In [57]: doc[2].pos_

Out[57]: 'VERB'

In [59]: doc[2].tag_

Out[59]: 'VB'

In [60]: spacy.explain('VB')

Out[60]: 'verb, base form'
```

```
In [61]: for word in doc:
             print(word.text,"------>", word.pos_,word.tag_,spacy.explain(word.tag_))

I ------> PRON PRP pronoun, personal
will ------> AUX MD verb, modal auxiliary
google ------> VERB VB verb, base form
about ------> ADP IN conjunction, subordinating or preposition
facebook ------> PROPN NNP noun, proper singular
```

Alternate :

```
import nltk
from nltk.tag import pos_tag
text = 'GeeksforGeeks is a very famous edutech company in the IT
industry.'
pos_tags = pos_tag(lowercase_tokens)
```

Named Entity Recognition (NER)

NER involves identifying and classifying named entities in text, such as people, organisations, and locations. This step is commonly used in various NLP tasks such as information extraction, machine translation, and question-answering

Github :
https://github.com/hrithikM86/Natural_Language_Processing/blob/main/6.%20Named%20Entity%20Recognition/nlp_tutorial_NER.ipynb

NLP Tutorial: Named Entity Recognition (NER)

In [3]:

```
import spacy
# Spacy is an open-source library for natural language processing (NLP) in Python, providing efficient tools for tokenizatio
# part-of-speech tagging, named entity recognition, and more.
```

In [6]:

```
nlp = spacy.load("en_core_web_sm")
# "en_core_web_sm" is a pre-trained English language model provided by the Spacy library, which includes tokenization,
# part-of-speech tagging, syntactic parsing, and named entity recognition capabilities.
nlp.pipe_names
```

Out[6]:

```
['tok2vec', 'tagger', 'parser', 'attribute_ruler', 'lemmatizer', 'ner']
```

In [11]:

```
nlp.pipe_names[5]
```

Out[11]:

```
'ner'
```

In [12]:

```
doc = nlp("Tesla Inc is going to acquire twitter for $45 billion")
for ent in doc.ents:
    print(ent.text, " | ", ent.label_, " | ", spacy.explain(ent.label_))
```

```
Tesla Inc  |  ORG  |  Companies, agencies, institutions, etc.
$45 billion  |  MONEY  |  Monetary values, including unit
```

In [13]:

```
from spacy import displacy

displacy.render(doc, style="ent")
```

Tesla Inc **ORG** is going to acquire twitter for $45 billion **MONEY**

List down all the entities

In [14]:

```
nlp.pipe_labels['ner']
```

Out[14]:

```
['CARDINAL',
 'DATE',
 'EVENT',
 'FAC',
 'GPE',
 'LANGUAGE',
 'LAW',
 'LOC',
 'MONEY',
 'NORP',
 'ORDINAL',
 'ORG',
 'PERCENT',
 'PERSON',
 'PRODUCT',
 'QUANTITY',
 'TIME',
 'WORK_OF_ART']
```

List of entities are also documented on this page: https://spacy.io/models/en

Setting custom entities

In [18]:
```
doc = nlp("Tesla is going to acquire Twitter for $45 billion")
for ent in doc.ents:
    print(ent.text, " | ", ent.label_)
```

```
Tesla  |  ORG
Twitter  |  PRODUCT
$45 billion  |  MONEY
```

In [19]:
```
s = doc[2:5]
s
```

Out[19]: going to acquire

In [20]:
```
type(s)
```

Out[20]: spacy.tokens.span.Span

In [21]:
```
from spacy.tokens import Span

s1 = Span(doc, 0, 1, label="ORG")
s2 = Span(doc, 5, 6, label="ORG")

doc.set_ents([s1, s2], default="unmodified")
```

In [22]:
```
for ent in doc.ents:
    print(ent.text, " | ", ent.label_)
```

```
Tesla  |  ORG
Twitter  |  ORG
$45 billion  |  MONEY
```

TEXT PREPROCESSING USING NLTK :

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer, WordNetLemmatizer
from nltk.tag import pos_tag
from nltk.chunk import ne_chunk
import string

# sample text to be preprocessed
text = 'GeeksforGeeks is a very famous edutech company in the IT
industry.'

# tokenize the text
tokens = word_tokenize(text)

# remove stop words
stop_words = set(stopwords.words('english'))
```

```python
filtered_tokens = [token for token in tokens if token.lower() not
in stop_words]

# perform stemming and lemmatization
stemmer = SnowballStemmer('english')
lemmatizer = WordNetLemmatizer()
stemmed_tokens = [stemmer.stem(token) for token in
filtered_tokens]
lemmatized_tokens = [lemmatizer.lemmatize(token) for token in
filtered_tokens]

# remove digits and punctuation
cleaned_tokens = [token for token in lemmatized_tokens
                  if not token.isdigit() and not token in
string.punctuation]

# convert all tokens to lowercase
lowercase_tokens = [token.lower() for token in cleaned_tokens]

# perform part-of-speech (POS) tagging
pos_tags = pos_tag(lowercase_tokens)

# perform named entity recognition (NER)
named_entities = ne_chunk(pos_tags)

# print the preprocessed text
print("Original text:", text)
print("Preprocessed tokens:", lowercase_tokens)
print("POS tags:", pos_tags)
print("Named entities:", named_entities)
```

4. Feature Engineering

In Feature Engineering, our main agenda is to represent the text in the numeric vector in such a way that the ML algorithm can understand the text attribute. In NLP this process of feature engineering is known as Text Representation or Text Vectorization

One Hot Encodings

One Hot Encoder for texts is a technique that converts categorical text data into binary vectors (1 if the word is present and 0 if it is not) to represent the presence or absence of each category. The vocabulary is formed by taking all the words. Then One Hot Encoding is implemented. If a test set has a word that is different from train set , OHE won't work .

```python
import nltk
# nltk.download('punkt') # Download 'punkt'
# from nltk if it's not downloaded
from nltk.tokenize import sent_tokenize
Text = """Geeks For Geeks.
         Geeks Learning Together.
         Geeks For Geeks is famous for DSA.
         Learning DSA"""
sentences = sent_tokenize(Text)
sentences = [sent.lower().replace(".", "") for sent in sentences]
print('Tokenized Sentences :', sentences)

# Create the vocabulary
vocab = {}
count = 0
for sent in sentences:
    for word in sent.split():
        if word not in vocab:
            count = count + 1
            vocab[word] = count
print('vocabulary :', vocab)

# One Hot Encoding
def OneHotEncoder(text):
    onehot_encoded = []
    for word in text.split():
        temp = [0]*len(vocab)
        if word in vocab:
            temp[vocab[word]-1] = 1
            onehot_encoded.append(temp)
    return onehot_encoded


# print('\n',sentences[0])
print('OneHotEncoded vector for sentence : "',
     sentences[0], '"is \n', OneHotEncoder(sentences[0]))
```

NOTE :
The `sent_tokenize` function from the NLTK library is used to split a text into individual sentences. It applies a rule-based approach to identify sentence boundaries based on punctuation and other language-specific cues. It helps in segmenting a paragraph or document into separate sentences for further analysis or processing.

Bag Of Words (BOW)

Bag of Words (BoW) is a text representation technique that converts text documents into a collection of unique words, disregarding grammar and word order, and representing each document as a vector of word frequencies

```
from sklearn.feature_extraction.text import CountVectorize
cv = CountVectorizer(max_features=3000)
X_train_bow = cv.fit_transform(X_train).toarray()
X_test_bow = cv.transform(X_test).toarray()
```

N-GRAMS

In Bag of Words, there is no consideration of the phrases or word order. Bag of n-gram tries to solve this problem by breaking text into chunks of n continuous words

```
from sklearn.feature_extraction.text import CountVectorize
cv = CountVectorizer(ngram_range=(1,2),max_features=5000)
X_train_bow = cv.fit_transform(X_train['review']).toarray()
X_test_bow = cv.transform(X_test['review']).toarray()
```

TF-IDF

Term Frequency – Inverse Document Frequency

In all the above techniques, Each word is treated equally. TF-IDF tries to quantify the importance of a given word relative to the other word in the corpus. it is mainly used in Information retrieval

Term Frequency (TF): TF measures how often a word occurs in the given document. it is the ratio of the number of occurrences of a term or word (t) in a given document (d) to the total number of terms in a given document (d)

$$\mathrm{TF}(t,d) = \frac{\text{(Number of occurrences of term t in document d)}}{\text{(Total number of terms in the document d)}}$$

Inverse document frequency (IDF): IDF measures the importance of the word across the corpus. it down the weight of the terms, which commonly occur in the corpus, and up the weight of rare terms.

$$\mathrm{IDF}(t) = \log_e \frac{\text{(Total number of documents in the corpus)}}{\text{(Number of documents with term t in corpus)}}$$

TF-IDF score is the product of TF and IDF

$$\text{TF-IDF Score} = TF \times IDF$$

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer()
X_train_tfidf = tfidf.fit_transform(X_train['review']).toarray()
X_test_tfidf = tfidf.transform(X_test['review'])
```

The above technique is not very good for complex tasks like Text Generation, Text summarization, etc. and they can't understand the contextual meaning of words

Neural Approach (Word embedding)

But in the neural approach or word embedding, we try to incorporate the contextual meaning of the words. Here each word is represented by real values as the vector of fixed dimensions.

1. Train our own embedding layer:

There are two ways to train our own word embedding vector :

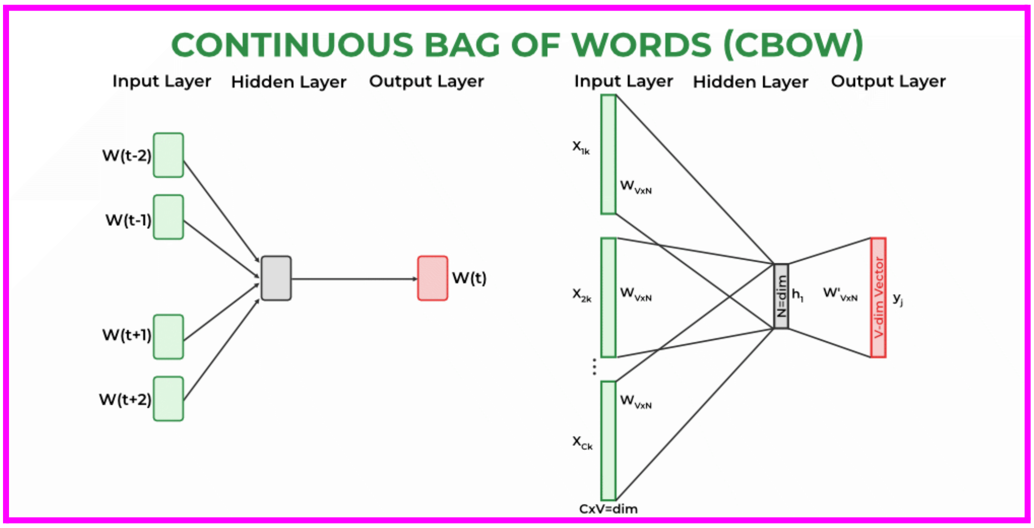
CBOW

CBOW (Continuous Bag of Words): In this case, we predict the centre word from the given set of context words i.e previous and afterwords of the centre word.

For example :

I am learning Natural Language Processing from GFG.

I am learning Natural ______?______ Processing from GFG.



The weights of the hidden layer are the word embeddings of the output word

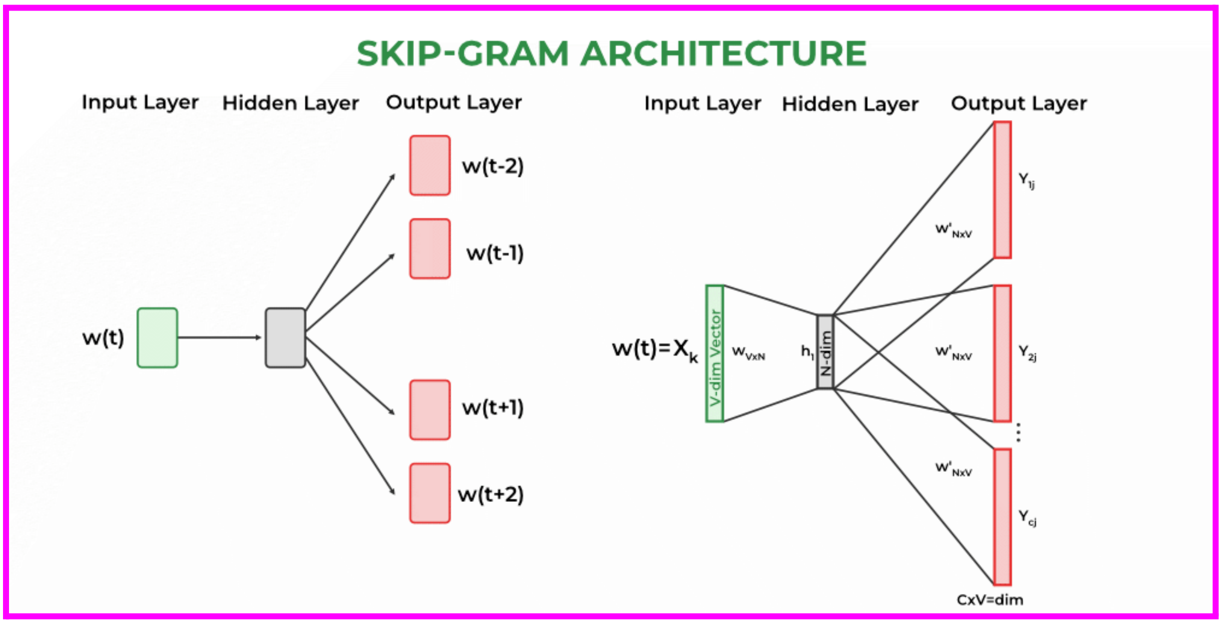
SkipGram

SkipGram: In this case, we predict the context word from the centre word.

For example :

I am learning Natural Language Processing from GFG.

I am __?___ _____?_____ Language ___?___ ____?____ GFG.



2. Pre-Trained Word Embeddings :

1. Word2vec by Google
2. GloVe by Stanford
3. fasttext by Facebook

Word2vec by Google

Github :
https://github.com/hrithikM86/Natural_Language_Processing/blob/main/4.%20word2vec.ipynb

```
import gensim.downloader as api

# load the pre-trained Word2Vec model
model = api.load('word2vec-google-news-300')

# define word pairs to compute similarity for
word_pairs = [('learn', 'learning'), ('india', 'indian'), ('fame',
'famous')]

# compute similarity for each pair of words
for pair in word_pairs:
    similarity = model.similarity(pair[0], pair[1])
    print(f"Similarity between '{pair[0]}' and '{pair[1]}' using
Word2Vec: {similarity:.3f}")
```

Complete Example :

Dataset :

Out[5]:

	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

First perform all the preprocessing :

Then perform Word2vec:

```
In [1]:  import gensim

In [13]: from nltk import sent_tokenize
         from gensim.utils import simple_preprocess

In [14]: story = []
         for doc in df['review']:
             raw_sent = sent_tokenize(doc)
             for sent in raw_sent:
                 story.append(simple_preprocess(sent))

In [15]: model = gensim.models.Word2Vec(
             window=10,
             min_count=20
         )

In [16]: model.build_vocab(story)

In [17]: model.train(story, total_examples=model.corpus_count, epochs=model.epochs)

Out[17]: (5875535, 6212140)

In [18]: len(model.wv.index_to_key)

Out[18]: 31845
```

The **simple_preprocess** function from the Gensim library is used to preprocess text data by performing basic tokenization and lowercasing. It takes a text document as input and returns a list of tokens (words) after applying the preprocessing steps.

The **sent_tokenize** function from the NLTK (Natural Language Toolkit) library is used to tokenize or split a text into individual sentences.

```
In [19]: def document_vector(doc):
             # remove out-of-vocabulary words
             doc = [word for word in doc.split() if word in model.wv.index_to_key]
             return np.mean(model.wv[doc], axis=0)

In [20]: document_vector(df['review'].values[0])

Out[20]: array([-0.13551651,  0.44236022,  0.19618301,  0.2243388 , -0.05095489,
                -0.5747312 ,  0.24217677,  0.9659006 , -0.35396117, -0.20832208,
                -0.2984078 , -0.4400784 ,  0.06137785,  0.0494826 ,  0.17676252,
                -0.12226024,  0.04628523, -0.3924497 , -0.03380001, -0.6410042 ,
                 0.04235367,  0.21378344,  0.05942454, -0.29909694, -0.3123422 ,
                -0.02930854, -0.28178704,  0.04482391, -0.32226902,  0.02552933,
                 0.3864304 ,  0.02701772,  0.14447899, -0.319503  , -0.16364487,
                 0.3655229 ,  0.09109591, -0.46899873, -0.21292035, -0.75319904,
                 0.09909733, -0.25865507,  0.06007399, -0.03030884,  0.48151824,
                -0.10978614, -0.25712606,  0.00799898,  0.05369115,  0.36769992,
                 0.07791721, -0.3666489 , -0.41765627, -0.11414295, -0.16897762,
                 0.24438773,  0.2116644 ,  0.08659696, -0.29066396,  0.08693315,
                 0.08008192,  0.12331163, -0.00459293, -0.11629856, -0.48819962,
                 0.23666628,  0.02842407,  0.15447702, -0.3748886 ,  0.23497324,
                -0.31478316,  0.09180178,  0.6065921 , -0.07799191,  0.36435187,
                 0.13831104,  0.02367809, -0.10589553, -0.52983063,  0.14002632,
                -0.3658514 ,  0.05524313, -0.4022981 ,  0.42953855, -0.14095813,
                -0.18778971, -0.06615544,  0.26965094,  0.3006101 ,  0.06881877,
                 0.2490601 ,  0.18089396,  0.01643727,  0.08703522,  0.656192  ,
                 0.32215416,  0.14057031, -0.1816499 , -0.06166815, -0.17208445],
               dtype=float32)
```

```python
from tqdm import tqdm

X = []
for doc in tqdm(df['review'].values):
    X.append(document_vector(doc))
```

Training a model :

```
In [29]: from sklearn.preprocessing import LabelEncoder
         encoder = LabelEncoder()

         y = encoder.fit_transform(df['sentiment'])

In [30]: y

Out[30]: array([1, 1, 1, ..., 0, 0, 1])

In [31]: from sklearn.model_selection import train_test_split
         X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=1)

In [32]: from sklearn.ensemble import RandomForestClassifier
         from sklearn.metrics import accuracy_score

In [33]: rf = RandomForestClassifier()
         rf.fit(X_train,y_train)
         y_pred = rf.predict(X_test)
         accuracy_score(y_test,y_pred)

Out[33]: 0.7681522283425137
```

5. Model Building , Evaluation , Deployment

GFG : https://www.geeksforgeeks.org/natural-language-processing-nlp-pipeline/

BERT : https://github.com/hrithikM86/Natural_Language_Processing/tree/main/BERT