Text Representation

—x—x—x—x—

↓ same

(Text vectorization)
or
(Feature Extraction)

Techniques:

- OHE
- BoW
- ngrams
- TF-IDF
- custom Features
- word Embedding's

*ONE HOT ENCODING: (OHE)

OHE in NLP represent words as binary vectors, where each word has unique index, & vector has one at the word's index & Zeros Elsewhere.

Example:

① Apple:
② Banana:
③ orang:

→ V=3 (vocab)

Apple → [1 0 0]
Banana → [0 1 0]
orange → [0 0 1]

Merits:

i) simplicity & intuitive
ii) work well with ML Algo.

Demerits:

i) Sparse Array & overfitting
ii) High Dimensionality.
iii) Equal Importance.
iv) ML Algo work with fix input shape.
v) OOV (out of vocabulary) issue

* Bag of words: (BoW)

↳ BoW is common technique in NLP for representing text data.
↳ It disregards the order & structure of words in doc, focusing only on word freq.

Example:

Doc 1 → 'The cat is on the mat'.
Doc 2 → 'The dog is chasing the cat'.

vocab = [The, cat, is, on, mat, dog, chasing]

BoW vectors:

Doc 1 → [2, 1, 1, 1, 1, 0, 0]
Doc 2 → [2, 1, 1, 0, 0, 1, 1]

Merits:

i) Simplicity.

ii) Applicable to various NLP tasks like sentiment analysis & doc classification.

Demerits:

i) Loss of sequence Info.

ii) High Dimensionality.

CountVectorizer Hyperparameter:

↳ Stopwords
↳ tokenizer
↳ max_df
↳ min_df
↳ max_features
↳ binary

Python Implementation:

```
text = ['the cat is on the mat', 'the dog is chasing
        the & cat']
from sklearn.feature_extraction.text import
 CountVectorizer
  cv = CountVectorizer()
  bow = cv.fit_transform(text)
   vocab = cv.vocabulary_
#Text into vectors
  bow[0].toarray()
```

*N-grams: Bag of N-grams

N-grams are sequential groups of n items (like words or chars.) in text. They are used in language processing to analyze pattern & relationships within text.

Example:

"The quick brown fox jumps over the lazy dog."

2-grams

the quick
quick brown
brown fox
fox jumps
jumps over
over the
the lazy
lazy dog

3-grams

the quick brown
quick brown fox
brown fox jumps
fox jumps over
jumps over the
over the lazy
the lazy dog

Benifits of N-grams:

i) Language modeling:
Capturing the context of words with in sequence, aiding tasks like speech Reco. & machine translation.

ii) Text Prediction:
Facilitate text prediction by analyzing patterns & predicting next word based on preceding ones, improving autocomplete & suggestion.

iii) Info. Retrieval:
N-grams enhance info. retrieval systems, helping match & Rank doc. based on the relevance of word sequence.

Advantage

Simplicity.

Feature Extraction.

Disadvantage

Limited context understanding.

Data sparsity.

Size & Storage.

TF-IDF : Term Freq. - Inverse Doc Freq.

(TF) – (IDF)

TF → measure how frequently a term appear in document.

IDF → measure of Importance of a term across a collection of documents.

If a word comes / present in a perticular document & rare in comparison of other documents, so that word is most important for that document. & vector value is higher.

Assign weightage to words

word

$$TF_{(t,d)} = \frac{(\text{No. of occurences of term } t \text{ in doc } d)}{(\text{Total No. of Terms in doc } d)}$$

$$IDF(t) = \log_e \frac{(\text{Total No. of Doc. in corpus})}{(\text{No. of DOC. with term } t \text{ in them})} + 1$$

(DF)

Example:

Doc1- machine learning algorithms enhance data analysis
Doc2- Data analysis is crucial for decision making
Doc3- Algorithms play a key role in machine learning.

(a: stopwords; in: stopwords)

Let's calculate TF-IDF for "machine" in each doc.

TF:

$TF(\text{"machine"}, Doc1) = \frac{1}{6}$

$TF(\text{"machine"}, Doc2) = 0$

$TF(\text{"machine"}, Doc3) = \frac{1}{7}$ ~~$\frac{1}{6}$~~

DF:

DF ("machine") = 2 (Appears in Doc1 & 3)

IDF:

$IDF(\text{"machine"}) = \log\left(\frac{3}{2}\right)$.

TF-IDF calculation:

$TF\text{-}IDF_{(D_1)} = \left(\frac{1}{6}\right) * \log(3/2)$

$TF\text{-}IDF(D_2) = 0 * \log(3/2)$

$TF\text{-}IDF(D_3) = \left(\frac{1}{6}\right) * \log(3/2)$

Text Data Representation:

One Hot Encoding Method: (OHE)

In [ ]:
```
from sklearn.preprocessing import OneHotEncoder

reshaped_text= [[text] for text in X2]
ohe= OneHotEncoder(sparse_output=True)
transformed_ohe_text=ohe.fit_transform(reshaped_text).toarray()
```

In [ ]:
```
transformed_ohe_text.shape
```

Out[ ]:
```
(40000, 39719)
```

In [ ]:
```
transformed_ohe_text[1]
```

Out[ ]:
```
array([0., 0., 0., ..., 0., 0., 0.])
```

Bag Of Words: (BoW)

In [ ]:
```
from sklearn.feature_extraction.text import CountVectorizer

cv= CountVectorizer()
transformed_bow= cv.fit_transform(X2).toarray()
```

In [ ]:
```
feature_name=cv.get_feature_names_out()
feature_name[:50]
```

```
Out[ ]: array(['aa', 'aaa', 'aaaa', 'aaaaaaaaaaaahhhhhhhhhhhhhh', 'aaaaaaaargh',
               'aaaaaaah', 'aaaaaaahhhhhhggg', 'aaaaagh', 'aaaaah', 'aaaaargh',
               'aaaaarrrrrgggggghhhhhhh', 'aaaaaw', 'aaaahhhhhh', 'aaaahhhhhhh',
               'aaaand', 'aaaggghhhhhhh', 'aaah', 'aaahhhhhhh', 'aaahthe',
               'aaargh', 'aachen', 'aada', 'aadha', 'aag', 'aage', 'aagh',
               'aaghh', 'aah', 'aahemy', 'aahhhh', 'aaila', 'aailiyah', 'aaip',
               'aaja', 'aajala', 'aak', 'aakash', 'aaker', 'aalcc', 'aaliyah',
               'aaliyahs', 'aalox', 'aames', 'aamess', 'aamilne', 'aamir',
               'aamirs', 'aamirso', 'aamr', 'aan'], dtype=object)
```

```
In [ ]: transformed_bow.shape
```

```
Out[ ]: (40000, 171932)
```

```
In [ ]: transformed_bow[201,:300]
```

```
Out[ ]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

N-Grams:

```
In [ ]: # Taking Samples of 5000 Reviews....
        X2= X2[:5000]
```

```
In [ ]: from sklearn.feature_extraction.text import CountVectorizer
```

```
cv_ngm= CountVectorizer(ngram_range=(2,2))
transformed_ngram= cv_ngm.fit_transform(X2).toarray()
```

In [ ]:
```
feature_name=cv_ngm.get_feature_names_out()
feature_name[:50]
```

Out[ ]:
```
array(['aa date', 'aa doctor', 'aaah friggin', 'aaargh fact',
       'aaip either', 'aakash go', 'aaliyah one', 'aamir ghulamone',
       'aamir khan', 'aamir prem', 'aamirs face', 'aamr toe',
       'aankhen fail', 'aardman animationswallace',
       'aardvarks unfortunately', 'aaron advice', 'aaron altman',
       'aaron carter', 'aaron carteryet', 'aaron eckhart', 'aaron great',
       'aaron michael', 'aaron seltzer', 'aaron sorkin',
       'aasize batteries', 'aasman ke', 'aatish kapadia', 'aavjo vhala',
       'ab akshay', 'aback shear', 'abandon alcatraz', 'abandon alone',
       'abandon baby', 'abandon beckon', 'abandon blackwell',
       'abandon build', 'abandon cars', 'abandon chess', 'abandon church',
       'abandon condemn', 'abandon didnt', 'abandon die',
       'abandon direction', 'abandon doctor', 'abandon eastwoods',
       'abandon egyptian', 'abandon family', 'abandon father',
       'abandon fiancee', 'abandon funeral'], dtype=object)
```

In [ ]:
```
transformed_ngram.shape
```

Out[ ]:
```
(5000, 421018)
```

In [ ]:
```
transformed_ngram[504,:700]
```

```
Out[ ]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

TF-IDF:

```
In [ ]: from sklearn.feature_extraction.text import TfidfVectorizer

tf_idf= TfidfVectorizer()
transformed_tf_idf= tf_idf.fit_transform(X2)
```

In [ ]:
```
# Assuming tf_idf is your TF-IDF vectorizer
feature_names = tf_idf.get_feature_names_out()

# Display the first 10 items in the vocabulary
print(feature_names[:50])
```

```
['aa' 'aaah' 'aaargh' 'aaip' 'aakash' 'aaliyah' 'aamir' 'aamirs' 'aamr'
 'aankhen' 'aardman' 'aardvarks' 'aaron' 'aasize' 'aasman' 'aatish'
 'aavjo' 'ab' 'aback' 'abandon' 'abandonment' 'abattoirs' 'abba' 'abbas'
 'abbey' 'abbot' 'abbott' 'abbyss' 'abc' 'abcs' 'abcsears' 'abdalla'
 'abdomen' 'abdoo' 'abduct' 'abduction' 'abductionman' 'abductions'
 'abductor' 'abdul' 'abdulrahman' 'abe' 'abecassis' 'abel' 'aberrations'
 'abet' 'abhor' 'abhorrent' 'abide' 'abigail']
```

In [ ]:
```
transformed_tf_idf.shape
```

Out[ ]:
```
(5000, 46198)
```

In [ ]:
```
transformed_tf_idf.toarray()[2,100:200]
```

Out[ ]:
```
array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.19940224, 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ])
```