Implementing TFIDF vectorizer

Task 1

In [77]:

```
corpus = [
     'this is the first document',
     'this document is the second document',
     'and this is the third one',
     'is this the first document ',
]
```

In [78]:

```
from collections import Counter
from tqdm import tqdm
from scipy.sparse import csr_matrix
import math as m
import operator
from sklearn.preprocessing import normalize
import numpy
```

In [79]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
vectorizer.fit(corpus)
skl_output = vectorizer.transform(corpus)
```

FIT FUNCTION FOR TFIDF

In [80]:

```
def fit(dataset):
    "CRAETING VOCABULARY OF UNIQUE WORDS FROM CORPUS"
    vocab = set()
    if isinstance(dataset, (list,)):
        for row in dataset:
            for word in row.split(" "):
                if len(word) < 2:
                    continue
                vocab.add(word)
        unique_words= sorted(list(vocab))
        vocab = {j:i for i,j in enumerate(unique_words)}
        return vocab
        print("you need to pass list of sentance")
```

In [81]:
```
vocab=fit(corpus)
print(vocab)
```

```
{'and': 0, 'document': 1, 'first': 2, 'is': 3, 'one': 4, 'second': 5,
'the': 6, 'third': 7, 'this': 8}
```

In [29]:
```
print(vectorizer.get_feature_names())
```

```
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'th
is']
```

- BOTH THE VOCABULARY ARE SAME

In [30]:
```
def tf(corpus,vocab):
    """FUNCTION TO CALCULATE TF(TERM FREQUENCY) VALUES OF DOCUMENT"""
    tf={}
    if isinstance(corpus, (list,)):
        for idx, row in enumerate(corpus):
                word_freq = dict(Counter(row.split()))
                c=sum(word_freq.values())
                for act in vocab.keys():
                    if len(act) < 2:
                        continue
                    if act in word_freq:
                        tf[act]=word_freq[act]/c

                    else:
                        tf[act]=0
    return(tf)
```

In [83]:
```
def idf(corpus,vocab):
    """FUNCTION OF CALCULATE IDF VALUES OF A CORPUS"""
    idf={}
    N=len(corpus)

    for act in vocab.keys():
        n=0
        for idx, row in enumerate((corpus)):
            if len(act) < 2:
                continue
            if act in row:
                n=n+1

        idf[act]=1+(m.log((1+N)/(1+float(n))))
    return(idf)
```

In [84]:
```
idf(corpus,vocab)
```

Out[84]:
```
{'and': 1.916290731874155,
 'document': 1.2231435513142097,
 'first': 1.5108256237659907,
 'is': 1.0,
 'one': 1.916290731874155,
 'second': 1.916290731874155,
 'the': 1.0,
 'third': 1.916290731874155,
 'this': 1.0}
```

In [39]:
```
print(vectorizer.idf_)
```

```
[1.91629073 1.22314355 1.51082562 1.         1.91629073 1.91629073
 1.         1.91629073 1.        ]
```

- BOTH IDF VALUES ARE SAME

```
In [82]: idfv=idf(corpus,vocab)
```

TRANSFORM FUNCTION

```
In [41]: def tranform(corpus,idfs):
             """TRANSFORM FUNCTION TO CALCULATE THE TFIDF VALUES FOR EACH DOCUME
         NT AND STORE IN SPARSE MATRIX"""
             rows=[]
             colums=[]
             values=[]
             tfidf={}
             if isinstance(corpus, (list,)):
                 for idx, row in enumerate(tqdm(corpus)):
                     lis=[]
                     lis.append(row) # for each document we are calling the func
         tion tf to calculate tf values for a particular document
                     tfval=tf(lis,vocab)
                     for word,value in idfs.items():
                         if len(word) < 2:
                             continue
                         tfidf[word]=idfv[word]*tfval[word]

                         if tfidf[word]!=0:
                             rows.append(idx)
                             colums.append(value)
                             values.append(tfidf[word])
                         l=csr_matrix((values, (rows,colums)), shape=(len(corpus
         ),len(vocab)))# creating the sparse matrix
                         k=normalize(l,norm='l2') # normalize the sparse matrix
         (l2 norm)
                 print(k[0])
                 print("_______________________________________________________"
         )
                 print(k[0].toarray()) #converting intro dense matrix
```

```
        print("shape of sparse matrix",k.shape)
```

In [42]:

```
#TFIDF VALUES OF FIRST DOCUMENT STORED IN SPARSE MATRIX AND CONVERTED I
NTO DENSE MATRIX
tranform(corpus,vocab)
```

```
100%|████████████████████████████████████████████████████████████████████
█████████████████| 4/4 [00:00<00:00, 166.77it/s]
```

```
  (0, 1)	0.4697913855799205
  (0, 2)	0.580285823684436
  (0, 3)	0.3840852409148149
  (0, 6)	0.3840852409148149
  (0, 8)	0.3840852409148149
________________________________________________________
[[0.         0.46979139 0.58028582 0.38408524 0.         0.
  0.38408524 0.         0.38408524]]
shape of sparse matrix (4, 9)
```

In [43]:

```
print(skl_output[0])

print("________________________________________________________")

print(skl_output[0].toarray())
```

```
  (0, 8)	0.38408524091481483
  (0, 6)	0.38408524091481483
  (0, 3)	0.38408524091481483
  (0, 2)	0.5802858236844359
  (0, 1)	0.46979138557992045
________________________________________________________
[[0.         0.46979139 0.58028582 0.38408524 0.         0.
  0.38408524 0.         0.38408524]]
```

- BOTH OUTPUT FOR FIRST DOCUMENT IS SAME

Task 2

In [57]:

```
import pickle
with open('cleaned_strings', 'rb') as f: #IMPORTING PICKLE FILE
    corpus = pickle.load(f)
print("Number of documents in corpus = ",len(corpus))
```

```
Number of documents in corpus =  746
```

In [46]:

```
def fit(dataset):
    "CRAETING VOCABULARY OF UNIQUE WORDS FROM CORPUS"
    vocab = set()
    if isinstance(dataset, (list,)):
        for row in dataset:
            for word in row.split(" "):
                if len(word) < 2:
                    continue
                vocab.add(word)
        unique_words= sorted(list(vocab))
        vocab = {j:i for i,j in enumerate(unique_words)}
        return vocab
        print("you need to pass list of sentance")
```

In [47]:

```
vocab=fit(corpus) # CREATING THE VOCAB OF ALL UNIQUE WORDS IN CORPUS
```

In [61]:

```
def idf(corpus,vocab):
    """FUNCTION OF CALCULATE IDF VALUES OF A CORPUS"""
    i=0
    idf={}
    idfsorted={}
    idf50={}
    vocab50={}
    N=len(corpus)
    for act in vocab.keys():
        n=0
        for idx, row in enumerate((corpus)):
            if len(act) < 2:
                continue
            if act in row:
                n=n+1
            idf[act]=1+(m.log((1+N)/(1+float(n))))

    for k in sorted(idf, key=idf.get, reverse=True):
            idfsorted[k]=idf[k]        #SORTING DICTIONARY OF IDF BASED
 ON KEY VALUES(IDF VALUES)
    z=Counter(idfsorted)
    top=z.most_common(50)            # STORING TOP 50 IDF VALUES
    for a,b in top:
        vocab50[a]=i               # CREATING VOCABULARY BASED ON TOP 5
0 IDF VALUES
        idf50[a]=b
        i=i+1
    return vocab50,idf50 # THESE FUNCTION RETURNS TOP 50 IDF VALUES AND
 WORDS CORRESPONDING TO THAT IDF VALUE
```

In [68]:

```
vocab50,idf50=idf(corpus,vocab)
```

In [69]:

```
print('*******IDF VALUES OF TOP 50 WORDS******')
print("\n")
print(idf50)
```

```
*******IDF VALUES OF TOP 50 WORDS******


{'aailiyah': 6.922918004572872, 'abandoned': 6.922918004572872, 'abroa
d': 6.922918004572872, 'abstruse': 6.922918004572872, 'academy': 6.9229
18004572872, 'accents': 6.922918004572872, 'accessible': 6.922918004572
872, 'acclaimed': 6.922918004572872, 'accolades': 6.922918004572872, 'a
ccurately': 6.922918004572872, 'achille': 6.922918004572872, 'ackerma
n': 6.922918004572872, 'adams': 6.922918004572872, 'added': 6.922918004
572872, 'admins': 6.922918004572872, 'admiration': 6.922918004572872,
'admitted': 6.922918004572872, 'adrift': 6.922918004572872, 'adventur
e': 6.922918004572872, 'aesthetically': 6.922918004572872, 'affected':
6.922918004572872, 'affleck': 6.922918004572872, 'afternoon': 6.9229180
04572872, 'agreed': 6.922918004572872, 'aimless': 6.922918004572872, 'a
ired': 6.922918004572872, 'akasha': 6.922918004572872, 'alert': 6.92291
8004572872, 'alike': 6.922918004572872, 'allison': 6.922918004572872,
'allowing': 6.922918004572872, 'alongside': 6.922918004572872, 'amateur
ish': 6.922918004572872, 'amazed': 6.922918004572872, 'amazingly': 6.92
2918004572872, 'amusing': 6.922918004572872, 'amust': 6.92291800457287
2, 'anatomist': 6.922918004572872, 'angela': 6.922918004572872, 'angeli
na': 6.922918004572872, 'angry': 6.922918004572872, 'anguish': 6.922918
004572872, 'angus': 6.922918004572872, 'animals': 6.922918004572872, 'a
nimated': 6.922918004572872, 'anita': 6.922918004572872, 'anniversary':
6.922918004572872, 'anthony': 6.922918004572872, 'antithesis': 6.922918
004572872, 'anyway': 6.922918004572872}
```

In [70]:

```
print('*******TOP 50 WORDS AFTER SORTED IDF VALUES******')
print("\n")
print(vocab50)
```

```
*******TOP 50 WORDS AFTER SORTED IDF VALUES******


{'aailiyah': 0, 'abandoned': 1, 'abroad': 2, 'abstruse': 3, 'academy':
4, 'accents': 5, 'accessible': 6, 'acclaimed': 7, 'accolades': 8, 'accu
rately': 9, 'achille': 10, 'ackerman': 11, 'adams': 12, 'added': 13, 'a
dmins': 14, 'admiration': 15, 'admitted': 16, 'adrift': 17, 'adventur
e': 18, 'aesthetically': 19, 'affected': 20, 'affleck': 21, 'afternoo
n': 22, 'agreed': 23, 'aimless': 24, 'aired': 25, 'akasha': 26, 'aler
t': 27, 'alike': 28, 'allison': 29, 'allowing': 30, 'alongside': 31, 'a
mateurish': 32, 'amazed': 33, 'amazingly': 34, 'amusing': 35, 'amust':
36, 'anatomist': 37, 'angela': 38, 'angelina': 39, 'angry': 40, 'anguis
h': 41, 'angus': 42, 'animals': 43, 'animated': 44, 'anita': 45, 'anniv
ersary': 46, 'anthony': 47, 'antithesis': 48, 'anyway': 49}
```

In [54]:

```
def tf(corpus,vocab):
    """FUNCTION TO CALCULATE TF(TERM FREQUENCY) VALUES OF DOCUMENT"""
    tf={}
    if isinstance(corpus, (list,)):
        for idx, row in enumerate(corpus):
                word_freq = dict(Counter(row.split()))
                c=sum(word_freq.values())
                for act in vocab.keys():
                    if len(act) < 2:
                        continue
                    if act in word_freq:
                        tf[act]=word_freq[act]/c

                    else:
                        tf[act]=0
    return(tf)
```

In [55]:

```
def tranform(corpus,idfs):
    """TRANSFORM FUNCTION TO CALCULATE THE TFIDF VALUES FOR EACH DOCUME
NT AND STORE IN SPARSE MATRIX"""
    rows=[]
    colums=[]
    values=[]
    tfidf={}
    if isinstance(corpus, (list,)):
        for idx, row in enumerate(tqdm(corpus)):
            lis=[]
            lis.append(row) # for each document we are calling the func
tion tf to calculate tf values for a particular document

            tfval=tf(lis,vocab50)  # WE ARE GIVING ONLY TOP 50 WORDS  F
OR TF CALCULATION
            for word,value in idf50.items():
                if len(word) < 2:
                    continue
                tfidf[word]=idf50[word]*tfval[word]

                if tfidf[word]!=0:
                    rows.append(idx)
                    colums.append(value)
                    values.append(tfidf[word])
                l=csr_matrix((values, (rows,colums)), shape=(len(corpus
),len(vocab50)))# creating the sparse matrix
                k=normalize(l,norm='l2',) # normalize the sparse matrix
(l2 norm)
        print(k[0])
        print("________________________________________________________"
)
        print(k[0].toarray()) #converting intro dense matrix
        print("shape of sparse matrix",k.shape)
```

In [56]:
```
tranform(corpus,vocab)
```

```
100%|██████████████████████████████████████████████████████████████
██████████████| 746/746 [00:19<00:00, 37.91it/s]
```

```
  (0, 6)        1.0
________________________________________________________
[[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.]]
shape of sparse matrix (746, 50)
```