Task 1

In [120]:
```
from collections import Counter
from tqdm import tqdm
from scipy.sparse import csr_matrix
import math
import operator
from sklearn.preprocessing import normalize
import numpy
```

In [121]:
```
corpus = [
     'this is the first document',
     'this document is the second document',
     'and this is the third one',
     'is this the first document',
]
```

In [122]:
```
def IDF(corpus, unique_words):
   idf_value={}
   N=len(corpus)
   for word in unique_words:
     count=0
     for document in corpus:
       if word in document.split():
         count=count+1
       idf_value[word]=(math.log((1+N)/(count+1)))+1
   return idf_value
```

In [123]:
```
def fit(dataset):
    unique_words = set()
    if isinstance(dataset, (list,)):
        for row in dataset:
            for word in row.split(" "):
                if len(word) < 2:
                    continue
                unique_words.add(word)
        unique_words = sorted(list(unique_words))
        vocab = {j:i for i,j in enumerate(unique_words)}
        IDF_cal=IDF(dataset,unique_words)

        return vocab,IDF_cal
    else:
        print("you need to pass list of sentance")

vocab,IDF_cal=fit(corpus)
```

In [124]:
```
print(vocab)
```

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

In [125]:
```
print(list(vocab.keys()))
```

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

In [126]:
```
print(IDF_cal)
```

```
{'and': 1.916290731874155, 'document': 1.2231435513142097, 'first': 1.5108256237659907, 'is': 1.0, 'one': 1.916290731
874155, 'second': 1.916290731874155, 'the': 1.0, 'third': 1.916290731874155, 'this': 1.0}
```

In [127]:
```
print(list(IDF_cal.values()))
```

```
[1.916290731874155, 1.2231435513142097, 1.5108256237659907, 1.0, 1.916290731874155, 1.916290731874155, 1.0, 1.9162907
31874155, 1.0]
```

Transform Method

In [128]:
```
def transform(dataset,vocabulary,idf_values):
     sparse_matrix= csr_matrix( (len(dataset), len(vocabulary)), dtype=np.float64)
     for row  in range(0,len(dataset)):
       word_freq=Counter(dataset[row].split())
       for word in dataset[row].split():
           if word in  list(vocabulary.keys()):
               tf_idf_value=(word_freq[word]/len(dataset[row].split()))*(idf_values[word])
               sparse_matrix[row,vocabulary[word]]=tf_idf_value
     output =normalize(sparse_matrix, norm='l2', axis=1, copy=True, return_norm=False)
     return output
output=transform(corpus,vocab,IDF_cal)
```

```
C:\Users\91888\Anaconda3\lib\site-packages\scipy\sparse\compressed.py:746: SparseEfficiencyWarning: Changing the spar
sity structure of a csr_matrix is expensive. lil_matrix is more efficient.
  SparseEfficiencyWarning)
```

In [129]:
```
print(output.shape)
```

```
(4, 9)
```

In [130]:
```
print(output[0])
```

```
  (0, 1)        0.4697913855799205
  (0, 2)        0.580285823684436
  (0, 3)        0.3840852409148149
  (0, 6)        0.3840852409148149
  (0, 8)        0.3840852409148149
```

Task 2

In [131]:
```
import pickle
with open('C:/Users/91888/Desktop/Assignment/TFIDF-assignment 3/cleaned_strings (1)', 'rb') as f:
    corpus = pickle.load(f)

# printing the length of the corpus loaded
print("Number of documents in corpus = ",len(corpus))
```

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

In [132]:
```
def IDF(corpus, unique_words):
   idf_value={}
   N=len(corpus)
   for word in unique_words:
     count=0
     for document in corpus:
       if word in document.split():
         count=count+1
       idf_value[word]=(math.log((1+N)/(count+1)))+1
   return idf_value
```

In [133]:
```
def fit(dataset):
    unique_words = set()
    if isinstance(dataset, (list,)):
        for row in dataset:
            for word in row.split(" "):
                if len(word) < 2:
                    continue
                unique_words.add(word)
        unique_words = sorted(list(unique_words))
        vocab = {j:i for i,j in enumerate(unique_words)}
        IDF_cal=IDF(dataset,unique_words)
        IDF_sort=dict(sorted(IDF_cal.items(), key=operator.itemgetter(1),reverse=True)[:50])
        top_words=set()
        for key in IDF_sort:
            top_words.add(key)
        top_words = sorted(list(top_words))
        new_vocab = {j:i for i,j in enumerate(top_words)}


        return vocab,IDF_cal,IDF_sort,new_vocab
    else:
        print("you need to pass list of sentance")
```

In [134]:
```
vocab,IDF_cal,IDF_sort,new_vocab=fit(corpus)
```

In [135]:
```
print(IDF_sort)
```

```
{'aailiyah': 6.922918004572872, 'abandoned': 6.922918004572872, 'abroad': 6.922918004572872, 'abstruse': 6.9229180045
72872, 'academy': 6.922918004572872, 'accents': 6.922918004572872, 'accessible': 6.922918004572872, 'acclaimed': 6.92
2918004572872, 'accolades': 6.922918004572872, 'accurate': 6.922918004572872, 'accurately': 6.922918004572872, 'achil
le': 6.922918004572872, 'ackerman': 6.922918004572872, 'actions': 6.922918004572872, 'adams': 6.922918004572872, 'ad
d': 6.922918004572872, 'added': 6.922918004572872, 'admins': 6.922918004572872, 'admiration': 6.922918004572872, 'adm
itted': 6.922918004572872, 'adrift': 6.922918004572872, 'adventure': 6.922918004572872, 'aesthetically': 6.9229180045
72872, 'affected': 6.922918004572872, 'affleck': 6.922918004572872, 'afternoon': 6.922918004572872, 'aged': 6.9229180
04572872, 'ages': 6.922918004572872, 'agree': 6.922918004572872, 'agreed': 6.922918004572872, 'aimless': 6.9229180045
72872, 'aired': 6.922918004572872, 'akasha': 6.922918004572872, 'akin': 6.922918004572872, 'alert': 6.92291800457287
2, 'alike': 6.922918004572872, 'allison': 6.922918004572872, 'allow': 6.922918004572872, 'allowing': 6.92291800457287
2, 'alongside': 6.922918004572872, 'amateurish': 6.922918004572872, 'amaze': 6.922918004572872, 'amazed': 6.922918004
572872, 'amazingly': 6.922918004572872, 'amusing': 6.922918004572872, 'amust': 6.922918004572872, 'anatomist': 6.9229
18004572872, 'angel': 6.922918004572872, 'angela': 6.922918004572872, 'angelina': 6.922918004572872}
```

In [137]:
```
print(new_vocab)
```

```
{'aailiyah': 0, 'abandoned': 1, 'abroad': 2, 'abstruse': 3, 'academy': 4, 'accents': 5, 'accessible': 6, 'acclaimed':
7, 'accolades': 8, 'accurate': 9, 'accurately': 10, 'achille': 11, 'ackerman': 12, 'actions': 13, 'adams': 14, 'add':
15, 'added': 16, 'admins': 17, 'admiration': 18, 'admitted': 19, 'adrift': 20, 'adventure': 21, 'aesthetically': 22,
'affected': 23, 'affleck': 24, 'afternoon': 25, 'aged': 26, 'ages': 27, 'agree': 28, 'agreed': 29, 'aimless': 30, 'ai
red': 31, 'akasha': 32, 'akin': 33, 'alert': 34, 'alike': 35, 'allison': 36, 'allow': 37, 'allowing': 38, 'alongsid
e': 39, 'amateurish': 40, 'amaze': 41, 'amazed': 42, 'amazingly': 43, 'amusing': 44, 'amust': 45, 'anatomist': 46, 'a
ngel': 47, 'angela': 48, 'angelina': 49}
```

In [166]:
```
def transform(dataset,vocabulary,idf_values):
     sparse_matrix= csr_matrix( (len(dataset), len(vocabulary)), dtype=np.float64)
     for row  in range(0,len(dataset)):
       word_freq=Counter(dataset[row].split())
       for word in dataset[row].split():
           if word in  list(vocabulary.keys()):
               tf_idf_value=(word_freq[word]/len(dataset[row].split()))*(idf_values[word])
               sparse_matrix[row,vocabulary[word]]=tf_idf_value
     output =normalize(sparse_matrix, norm='l2', axis=1, copy=True, return_norm=False)
     return output
output=transform(corpus,new_vocab,IDF_sort)
```

```
C:\Users\91888\Anaconda3\lib\site-packages\scipy\sparse\compressed.py:746: SparseEfficiencyWarning: Changing the spar
sity structure of a csr_matrix is expensive. lil_matrix is more efficient.
  SparseEfficiencyWarning)
```

In [167]:
```
A=output[0]
print(A)
```

```
  (0, 30)       1.0
```

In [163]:
```
A.todense()
```

Out[163]:
```
matrix([[0., 0., 0., 0., 0., 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.]])
```