Task 1 : Custom Implementation of TFIDF Vectorizer.

In [1]:

```
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
from tqdm import tqdm
import os
from collections import Counter
from scipy.sparse import csr_matrix
import numpy as np
import math as m

from tqdm import tqdm


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)}
        #Creating a list of number of documents each unique word is :
        doc_num=[]
        for i in range (0,len(unique_words)):
            count=0
            for j in range (0,len(dataset)):
                for k in range (0,len(dataset[j].split())):

                    if unique_words[i]==dataset[j].split()[k]:
                        count=count+1
                        break
            doc_num.append(count)



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


def transform(dataset,vocab,num):
    rows = []
    columns = []
    values = []
    #Computing idf values for each unique word
    idf=[]
    nume=1+len(dataset)
    for g in range (0,len(num)):

        den=1+num[g]
        idf_val=1+m.log(nume/den)
        idf.append(idf_val)
    print("The custom implemented idf values are : ",idf)
    if isinstance(dataset, (list,)):

        for idx, row in enumerate(tqdm(dataset)):
            word_freq = dict(Counter(row.split()))

            for word, freq in word_freq.items():

                if len(word) < 2:
                    continue
                #Computing tf value
                tf=freq/len(dataset[idx].split())
                col_index = vocab.get(word, -1)
                if col_index !=-1:
                    rows.append(idx)
                    columns.append(col_index)
                    #getting Tf*idf
                    values.append(tf*idf[col_index])


        return csr_matrix((values, (rows,columns)), shape=(len(dataset),len(vocab)))
    else:
        print("you need to pass list of strings")
dataset = [
     'this is the first document',
     'this document is the second document',
     'and this is the third one',
     'is this the first document',
]
from sklearn import preprocessing
vocab,num=fit(dataset)
print(list(vocab.keys()))
X=transform(dataset,vocab,num).toarray()
Y=csr_matrix(preprocessing.normalize(X,norm='l2'))
```

```
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
The custom implemented idf values are :  [1.916290731874155, 1.2231435513142
097, 1.5108256237659907, 1.0, 1.916290731874155, 1.916290731874155, 1.0, 1.9
16290731874155, 1.0]

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

The feature names and vocab have the same unique words.

In [2]:

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

print(vectorizer.get_feature_names())
```

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

The custom idf values and TfidfVectorizer values are same .

In [3]:

```
print(vectorizer.idf_)
```

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

The tfidf values for a particular document(here doc=0) is obtained and verified

In [4]:

```
print(Y[0,:])
```

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

In [5]:

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

```
  (0, 8)	0.38408524091481483
  (0, 6)	0.38408524091481483
  (0, 3)	0.38408524091481483
  (0, 2)	0.5802858236844359
  (0, 1)	0.46979138557992045
```

Printing dense matrix for a particular doc (here doc=0).

In [6]:

```
print(Y[0,:].toarray())
```

```
[[0.         0.46979139 0.58028582 0.38408524 0.         0.
  0.38408524 0.         0.38408524]]
```

Task 2 :

In [27]:

```
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
from tqdm import tqdm
import os
from collections import Counter
from scipy.sparse import csr_matrix
import numpy as np
import math as m

from tqdm import tqdm


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))
        #Computing for each unique word : in how many documents does the word occur
        doc_num=[]
        for i in range (0,len(unique_words)):
            count=0
            for j in range (0,len(dataset)):
                for k in range (0,len(dataset[j].split())):

                    if unique_words[i]==dataset[j].split()[k]:
                        count=count+1
                        break
            doc_num.append(count)
        #Computing idf values
        idf=[]
        nume=1+len(dataset)
        for g in range (0,len(doc_num)):
            den=1+doc_num[g]
            idf_val=1+m.log(nume/den)
            idf.append(idf_val)
        #making a dictionary of unique words with idf values
        vocab = {j:idf[i] for i,j in enumerate(unique_words)}
        #calculating words with highest idf value
        d=Counter(vocab)
        #getting top 50 idf values
        voc=dict(d.most_common(50))
        idf=sorted(idf)
        idf.reverse()
        idf=idf[0:50]
        word=voc.keys()
        #creating a dictionary of top 50 idf words for thetransform function
        voc1={k:l for l,k in enumerate(word)}



        return voc1,doc_num,idf
    else:
        print("you need to pass list of sentance")
```

```python
def transform(dataset,vocab,num,idf):
    rows = []
    columns = []
    values = []

    if isinstance(dataset, (list,)):

        for idx, row in enumerate(tqdm(dataset)):
            word_freq = dict(Counter(row.split()))

            for word, freq in word_freq.items():

                if len(word) < 2:
                    continue
                tf=freq/len(dataset[idx].split())
                col_index = vocab.get(word, -1)
                if col_index !=-1:
                    rows.append(idx)
                    columns.append(col_index)
                    values.append(tf*idf[col_index])


        return csr_matrix((values, (rows,columns)), shape=(len(dataset),len(vocab)))
    else:
        print("you need to pass list of strings")
import pickle
with open('cleaned_strings', 'rb') as f:
    dataset = pickle.load(f)

vocab,num,idf=fit(dataset)
X=transform(dataset,vocab,num,idf)
Y=csr_matrix(preprocessing.normalize(X,norm='l2'))
#top 50 idf values
print(idf)
#top 50 words with idf values
print(vocab)
print(Y[0,:].toarray())
```

```
100%|████████████████████████████████████████████████████████████████████
██████| 746/746 [00:00<00:00, 12896.77it/s]

[6.922918004572872, 6.922918004572872, 6.922918004572872, 6.92291800457287
2, 6.922918004572872, 6.922918004572872, 6.922918004572872, 6.922918004572
872, 6.922918004572872, 6.922918004572872, 6.922918004572872, 6.9229180045
72872, 6.922918004572872, 6.922918004572872, 6.922918004572872, 6.92291800
4572872, 6.922918004572872, 6.922918004572872, 6.922918004572872, 6.922918
004572872, 6.922918004572872, 6.922918004572872, 6.922918004572872, 6.9229
18004572872, 6.922918004572872, 6.922918004572872, 6.922918004572872, 6.92
2918004572872, 6.922918004572872, 6.922918004572872, 6.922918004572872, 6.
922918004572872, 6.922918004572872, 6.922918004572872, 6.922918004572872,
6.922918004572872, 6.922918004572872, 6.922918004572872, 6.92291800457287
2, 6.922918004572872, 6.922918004572872, 6.922918004572872, 6.922918004572
872, 6.922918004572872, 6.922918004572872, 6.922918004572872, 6.9229180045
72872, 6.922918004572872, 6.922918004572872, 6.922918004572872]
{'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, 'adam
s': 14, 'add': 15, 'added': 16, 'admins': 17, 'admiration': 18, 'admitte
d': 19, 'adrift': 20, 'adventure': 21, 'aesthetically': 22, 'affected': 2
3, 'affleck': 24, 'afternoon': 25, 'aged': 26, 'ages': 27, 'agree': 28, 'a
greed': 29, 'aimless': 30, 'aired': 31, 'akasha': 32, 'akin': 33, 'alert':
34, 'alike': 35, 'allison': 36, 'allow': 37, 'allowing': 38, 'alongside':
39, 'amateurish': 40, 'amaze': 41, 'amazed': 42, 'amazingly': 43, 'amusin
g': 44, 'amust': 45, 'anatomist': 46, 'angel': 47, 'angela': 48, 'angelin
a': 49}
[[0. 0. 0. 0. 0. 0. 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.]]
```

In [28]:

```
print(Y[0,:])
```

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

In [ ]: