Open in Colab
(https://colab.research.google.com/github/poojashah19/Data-Science/blob/main/Assignment%205/TFIDF_vectorizer.ipynb)

Task-1

In [23]:
```
corpus = [
     'this is the first document',
     'this document is the second document',
     'and this is the third one',
     'is this the first document',
]
```

In [24]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import preprocessing
from pandas import DataFrame

def document_matrix(list, vectorizer):
    doc_matrix = vectorizer.fit_transform(list)
    return DataFrame(doc_matrix.toarray(), columns = vectorizer.get_feature_na
mes())

count_vectorizer = CountVectorizer()
tfidf_vectorizer = TfidfVectorizer()
```

In [25]:
```
## Prints the number of words appear in a particular document
count_output = document_matrix(corpus, count_vectorizer)
print(count_output)
```

```
   and  document  first  is  one  second  the  third  this
0    0         1      1   1    0       0    1      0     1
1    0         2      0   1    0       1    1      0     1
2    1         0      0   1    1       0    1      1     1
3    0         1      1   1    0       0    1      0     1
```

In [26]:
```
## Prints the tfidf value of words in a particular document
tfidf_output = document_matrix(corpus, tfidf_vectorizer)
print(tfidf_output)
```

```
        and  document     first  ...       the     third      this
0  0.000000  0.469791  0.580286  ...  0.384085  0.000000  0.384085
1  0.000000  0.687624  0.000000  ...  0.281089  0.000000  0.281089
2  0.511849  0.000000  0.000000  ...  0.267104  0.511849  0.267104
3  0.000000  0.469791  0.580286  ...  0.384085  0.000000  0.384085

[4 rows x 9 columns]
```

```
In [27]: print(tfidf_vectorizer.get_feature_names())

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

```
In [28]: print(tfidf_vectorizer.idf_)

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

```
In [29]: tfidf_output.shape

Out[29]: (4, 9)
```

```
In [30]: skl_output = tfidf_vectorizer.transform(corpus)
         print(skl_output)

           (0, 8)        0.38408524091481483
           (0, 6)        0.38408524091481483
           (0, 3)        0.38408524091481483
           (0, 2)        0.5802858236844359
           (0, 1)        0.46979138557992045
           (1, 8)        0.281088674033753
           (1, 6)        0.281088674033753
           (1, 5)        0.5386476208856763
           (1, 3)        0.281088674033753
           (1, 1)        0.6876235979836938
           (2, 8)        0.267103787642168
           (2, 7)        0.511848512707169
           (2, 6)        0.267103787642168
           (2, 4)        0.511848512707169
           (2, 3)        0.267103787642168
           (2, 0)        0.511848512707169
           (3, 8)        0.38408524091481483
           (3, 6)        0.38408524091481483
           (3, 3)        0.38408524091481483
           (3, 2)        0.5802858236844359
           (3, 1)        0.46979138557992045
```

```
In [31]: print(skl_output[3])

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

```
In [32]: print(skl_output[0].toarray())

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

In [33]:

```
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 as np
```

In [34]:

```
def get_unique_words(data):
  unique_words = set()

  if isinstance(data, (list,)):
    for row in data:
      for word in row.split(' '):
        if(len(word) < 2):
          continue
        unique_words.add(word)

    unique_words = sorted(list(unique_words))
    return unique_words
  else:
    print('pass list of sentences')


def get_vocab(unique_words):
  vocab = {j:i for i,j in enumerate(unique_words)}
  return vocab


def transform(corpus, vocab):
  rows = []
  columns = []
  values = []

  if isinstance(corpus, (list,)):
    for index, row in enumerate(tqdm(corpus)):
      word_freq = dict(Counter(row.split()))
      for word, freq in word_freq.items():
        if len(word) < 2:
          continue

        col_index = vocab.get(word, -1)
        if col_index != -1:
          rows.append(index)
          columns.append(col_index)
          values.append(freq)
    return csr_matrix((values, (rows, columns)), shape = (len(corpus), len(voc
ab)))
  else:
    print('pass a list of strings')


def get_freq(corpus, unique_words):
  flattened = [val for sublist in corpus for val in sublist.split(' ')]
  freq = {}
  for word in unique_words:
    freq[word] = flattened.count(word)
  return freq


def find_in_str(str, word):
  str_list = str.split(' ')
  for i in range(len(str_list)):
    if(word == str_list[i]):
      return True
  return False


def compute_tfidf(corpus, unique_words, transform_output):
  rows = []
  columns = []
  tf = []
  idf = []
  values = []

  for i in range(len(corpus)):
    count = 0

    for j in range(len(unique_words)):
        temp = transform_output[i][j]
        if(temp > 0):
           count += temp
    for j in range(len(unique_words)):
        temp = transform_output[i][j]
        if(temp > 0):
            tf_value = temp / count
            idf_value = math.log( (len(corpus) + 1)/( float(get_idf(corpus, un
ique_words[j] ) + 1)) ) + 1
            rows.append(i)
            columns.append(j)
            values.append(tf_value * idf_value)
  return csr_matrix((values, (rows, columns)), shape = (len(corpus), len(uniqu
e_words)))


def get_idf(corpus, word):
  count = 0
  for j in range(len(corpus)):
    if(find_in_str(corpus[j], word)):
      count += 1
  return count
```

In [35]:

```
unique_words = get_unique_words(corpus)
vocab = get_vocab(unique_words)
frequency_of_words = get_freq(corpus, unique_words)
sparse_matrix = transform(corpus, vocab)
transform_output = transform(corpus, vocab).toarray()
print("\n")
# print(unique_words)
# print(vocab)
# print(frequency_of_words)
# print(sparse_matrix)
# print(transform_output)
tf_idf = compute_tfidf(corpus, unique_words, transform_output)
print(round(normalize(tf_idf, norm = 'l2'), 6))
```

```
100%|██████████| 4/4 [00:00<00:00, 7073.03it/s]
100%|██████████| 4/4 [00:00<00:00, 5344.76it/s]


  (0, 1)	0.469791
  (0, 2)	0.580286
  (0, 3)	0.384085
  (0, 6)	0.384085
  (0, 8)	0.384085
  (1, 1)	0.687624
  (1, 3)	0.281089
  (1, 5)	0.538648
  (1, 6)	0.281089
  (1, 8)	0.281089
  (2, 0)	0.511849
  (2, 3)	0.267104
  (2, 4)	0.511849
  (2, 6)	0.267104
  (2, 7)	0.511849
  (2, 8)	0.267104
  (3, 1)	0.469791
  (3, 2)	0.580286
  (3, 3)	0.384085
  (3, 6)	0.384085
  (3, 8)	0.384085
```

In [36]:

```
tfidf_output = document_matrix(corpus, tfidf_vectorizer)
print(tfidf_output)
```

```
        and  document     first  ...       the     third      this
0  0.000000  0.469791  0.580286  ...  0.384085  0.000000  0.384085
1  0.000000  0.687624  0.000000  ...  0.281089  0.000000  0.281089
2  0.511849  0.000000  0.000000  ...  0.267104  0.511849  0.267104
3  0.000000  0.469791  0.580286  ...  0.384085  0.000000  0.384085

[4 rows x 9 columns]
```

Observation:

- The list of unique words and their frequencies in entire document is same as the one calculated using TfidfVectorizer
- The transform matrix that contains unique words is same as get_feature_names of scikit learn TfidfVectorizer
- The transform output matrix is same as count_output that we calculated previously using scikit learn CountVectorizer.
- Shape of the transform matrix matches with the one calculated using TfidfVectorizer transform method.
- IDF values of all unique words from the entire document matches the values that were counted using scikit learn TfidfVectorizer.
- IDF_ values calculated by multiplying TF*IDF values individually, matches with the values calculated using TfidfVectorizer._idf

Task-2

In [37]:

```
import pickle
with open('cleaned_strings', 'rb') as f:
  corpus = pickle.load(f)

print("Number of documents in corpus = ",len(corpus))

tfidf_output = document_matrix(corpus, tfidf_vectorizer)
data = tfidf_output[:5]
```

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

In [38]:

```
def compute_topidf(corpus, unique_words):
    vocab = []
    idf = {}
    idf_50 = {}

    for col in corpus.columns:
      idf_value = math.log( (len(corpus) + 1)/( len( corpus[(corpus[col] > 0)]
) + 1 ) ) + 1
      idf[col] = idf_value
    idf_50 = { k:v for k, v in sorted( idf.items(), key = lambda item: item[1
], reverse=True )[:50] }

    vocab = {j:i for i,j in enumerate( list( idf_50.keys() ) )}
    return idf_50, vocab
```

In [39]:

```
unique_words = data.columns
idf_50, vocab = compute_topidf(data, unique_words)
print(vocab)
print(idf_50)
```

```
{'aailiyah': 0, 'abandoned': 1, 'ability': 2, 'abroad': 3, 'absolutely': 4,
'abstruse': 5, 'abysmal': 6, 'academy': 7, 'accents': 8, 'accessible': 9, 'ac
claimed': 10, 'accolades': 11, 'accurate': 12, 'accurately': 13, 'accused': 1
4, 'achievement': 15, 'achille': 16, 'ackerman': 17, 'act': 18, 'acted': 19,
'action': 20, 'actions': 21, 'actor': 22, 'actors': 23, 'actress': 24, 'actre
sses': 25, 'actually': 26, 'adams': 27, 'adaptation': 28, 'add': 29, 'added':
30, 'addition': 31, 'admins': 32, 'admiration': 33, 'admitted': 34, 'adorabl
e': 35, 'adrift': 36, 'adventure': 37, 'advise': 38, 'aerial': 39, 'aesthetic
ally': 40, 'affected': 41, 'affleck': 42, 'afraid': 43, 'africa': 44, 'aftern
oon': 45, 'age': 46, 'aged': 47, 'ages': 48, 'ago': 49}
{'aailiyah': 2.791759469228055, 'abandoned': 2.791759469228055, 'ability': 2.
791759469228055, 'abroad': 2.791759469228055, 'absolutely': 2.79175946922805
5, 'abstruse': 2.791759469228055, 'abysmal': 2.791759469228055, 'academy': 2.
791759469228055, 'accents': 2.791759469228055, 'accessible': 2.79175946922805
5, 'acclaimed': 2.791759469228055, 'accolades': 2.791759469228055, 'accurat
e': 2.791759469228055, 'accurately': 2.791759469228055, 'accused': 2.79175946
9228055, 'achievement': 2.791759469228055, 'achille': 2.791759469228055, 'ack
erman': 2.791759469228055, 'act': 2.791759469228055, 'acted': 2.7917594692280
55, 'action': 2.791759469228055, 'actions': 2.791759469228055, 'actor': 2.791
759469228055, 'actors': 2.791759469228055, 'actress': 2.791759469228055, 'act
resses': 2.791759469228055, 'actually': 2.791759469228055, 'adams': 2.7917594
69228055, 'adaptation': 2.791759469228055, 'add': 2.791759469228055, 'added':
2.791759469228055, 'addition': 2.791759469228055, 'admins': 2.79175946922805
5, 'admiration': 2.791759469228055, 'admitted': 2.791759469228055, 'adorabl
e': 2.791759469228055, 'adrift': 2.791759469228055, 'adventure': 2.7917594692
28055, 'advise': 2.791759469228055, 'aerial': 2.791759469228055, 'aesthetical
ly': 2.791759469228055, 'affected': 2.791759469228055, 'affleck': 2.791759469
228055, 'afraid': 2.791759469228055, 'africa': 2.791759469228055, 'afternoo
n': 2.791759469228055, 'age': 2.791759469228055, 'aged': 2.791759469228055,
'ages': 2.791759469228055, 'ago': 2.791759469228055}
```