Creating TF-IDFVectorizer

In [36]:
```
import warnings
warnings.filterwarnings("ignore")

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


def fit(dataset, top_50_idf = False):
    if isinstance(dataset, (list,)):
        unique_words = set()
        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)}
        #features = unique_words

        #Calculating idf of features
        dict_IDF = calculateIDF(dataset, vocab)

        #Sorting IDF in reverse and vocab as per the top 50 idf values
        if(top_50_idf):
            dict_IDF = dict(sorted(dict_IDF.items(), key=lambda kv: kv[1], reverse=True)[:50])

            vocab = {word : idx for idx ,word in enumerate(dict_IDF.keys())}

        return vocab, dict_IDF
    else:
        print("you need to pass list of sentence")




def calculateTF(word_frequency, document_len):
    return word_frequency/document_len


def calculateIDF(dataset, vocab):

    corpus_len = len(dataset)

    word_in_no_of_doc = {word : 0 for word, frequency in vocab.items()}

    for document in dataset:
        for word in vocab:
            if word in document:
                word_in_no_of_doc[word] = int(word_in_no_of_doc[word]) + 1


    dict_IDF = word_in_no_of_doc

    for word, occurences in word_in_no_of_doc.items():
        dict_IDF[word] = 1 + math.log( (1+ corpus_len )/ (1 + occurences))


    return dict_IDF




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

    if isinstance(dataset, (list,)):

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

            document_len = len(row)

            for word in row.split():
                if len(word) < 2:
                    continue

                col_index = vocab.get(word, -1)

                if col_index !=-1:
                    rows.append(idx)
                    columns.append(col_index)

                    tf = calculateTF(word_freq[word], document_len)
                    idf = dict_IDF[word]


                    tfidf = tf * idf

                    values.append(tfidf)

        csr_mat = csr_matrix((values, (rows,columns)), shape=(len(dataset),len(vocab)))

        normlized_matrix = normalize(csr_mat, norm='l2')


        return normlized_matrix

    else:
        print("you need to pass list of strings")
```

Task 1

In [41]:
```
corpus = [
     'this is the first document',
     'this document is the second document',
     'and this is the third one',
     'is this the first document',
]

vocab, dictIDF = fit(corpus)
print(list(vocab.keys()))

tfidf = transform(corpus, vocab, dictIDF)
print(tfidf)
```

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

```
100%|████████████████████████████████████████████████████████████████████████████| 4/4 [00:00<00:00, 3996.48it/s]
```

```
  (0, 1)        0.469791385579920 45
  (0, 2)        0.580285823684435 9
  (0, 3)        0.384085240914814 9
  (0, 6)        0.384085240914814 9
  (0, 8)        0.384085240914814 9
  (1, 1)        0.884320393165671 9
  (1, 3)        0.180747466684411 58
  (1, 5)        0.346364695217070 53
  (1, 6)        0.180747466684411 58
  (1, 8)        0.180747466684411 58
  (2, 0)        0.511848512707169
  (2, 3)        0.267103787642168
  (2, 4)        0.511848512707169
  (2, 6)        0.267103787642168
  (2, 7)        0.511848512707169
  (2, 8)        0.267103787642168
  (3, 1)        0.469791385579920 45
  (3, 2)        0.580285823684435 9
  (3, 3)        0.384085240914814 9
  (3, 6)        0.384085240914814 9
  (3, 8)        0.384085240914814 9
```

Task 2

In [42]:
```
import pickle
with open('cleaned_strings', 'rb') as f:
    corpus = pickle.load(f)


vocab, dict_IDF = fit(corpus, True)
print(list(vocab.keys()))

tfidf = transform(corpus, vocab, dict_IDF)
print(tfidf)
```

```
['aaliyah', 'abandoned', 'abroad', 'abstruse', 'academy', 'accents', 'accessible', 'acclaimed',
'accolades', 'accurately', 'achille', 'ackerman', 'adams', 'added', 'admins', 'admiration', 'adm
itted', 'adrift', 'adventure', 'aesthetically', 'affected', 'affleck', 'afternoon', 'agreed', 'a
imless', 'aired', 'akasha', 'alert', 'alike', 'allison', 'allowing', 'alongside', 'amateurish',
'amazed', 'amazingly', 'amusing', 'amust', 'anatomist', 'angela', 'angelina', 'angry', 'anguis
h', 'angus', 'animals', 'animated', 'anita', 'anniversary', 'anthony', 'antithesis', 'anyway']
```

```
100%|████████████████████████████████████████████████████████████████████████████| 746/746 [00:00<00:00, 74667.72it/s]
```

```
  (0, 24)       1.0
  (19, 43)      1.0
  (68, 21)      1.0
  (72, 23)      1.0
  (74, 25)      1.0
  (89, 47)      1.0
  (135, 3)      0.3779644730092272
  (135, 9)      0.3779644730092272
  (135, 15)     0.3779644730092272
  (135, 17)     0.3779644730092272
  (135, 29)     0.3779644730092272
  (135, 32)     0.3779644730092272
  (135, 40)     0.3779644730092272
  (176, 39)     1.0
  (192, 18)     1.0
  (193, 20)     1.0
  (216, 2)      1.0
  (225, 16)     1.0
  (227, 14)     1.0
  (241, 35)     1.0
  (270, 1)      1.0
  (290, 22)     1.0
  (341, 34)     1.0
  (344, 33)     1.0
  (348, 8)      1.0
  (409, 5)      1.0
  (430, 31)     1.0
  (457, 36)     1.0
  (461, 4)      0.7071067811865476
  (461, 44)     0.7071067811865476
  (465, 30)     1.0
  (475, 28)     1.0
  (493, 6)      1.0
  (500, 38)     1.0
  (544, 41)     1.0
  (548, 0)      0.7071067811865476
  (548, 26)     0.7071067811865476
  (608, 12)     1.0
  (612, 10)     1.0
  (620, 37)     0.7071067811865476
  (620, 42)     0.7071067811865476
  (632, 7)      1.0
  (644, 11)     0.5773502691896258
  (644, 45)     0.5773502691896258
  (644, 46)     0.5773502691896258
  (667, 19)     1.0
  (691, 27)     1.0
  (699, 48)     1.0
  (722, 13)     1.0
  (735, 49)     1.0
```