In [42]:

```
# Write your code here.
# Make sure its well documented and readble with appropriate comments.
# Compare your results with the above sklearn tfidf vectorizer
# You are not supposed to use any other library apart from the ones given below

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 [43]:

```
# it accepts only list of sentances
def fit(dataset):
    unique_words = set() # at first we will initialize an empty set
    idf = [] # idf value of each unique word
    # check if its list type or not
    if isinstance(dataset, (list,)):
        for row in dataset: # for each review in the dataset
            for word in row.split(" "): # for each word in the review. #split method co
                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)}

        # calculates the IDF for each word in vocabulary
        for word in vocab.keys():
            count = 0
            for doc in dataset:
                if word in doc.split(" "):
                    count+=1
            k = math.log((len(dataset)+1)/(count+1))+1
            idf.append(k)
        return vocab,idf
    else:
        print("you need to pass list of sentance")
```

In [44]:

```
def transform(corpus,vocabulary,IDF):
    rows = []
    columns = []
    TF = []
    values = []

    #Here we create word frequency matrix and total number of words in document of Corp
    if(isinstance(corpus,list)):
        for row,doc in enumerate(corpus):
            word_freq = dict(Counter(doc.split(" ")))
            total_words = 0
            for word,freq in word_freq.items():
                if len(word) < 2:
                    continue
                col = vocabulary.get(word,-1)
                total_words = total_words + freq
                if col != -1 :
                    rows.append(row)
                    columns.append(col)
                    values.append(freq)
            TF.append(total_words)
    Freq_Matrix = csr_matrix((values,(rows,columns)),(len(corpus),len(vocabulary)),floa

    #store the TF of each word in document
    for i in range(len(corpus)):
        for j in range(len(vocabulary)):
            if TF[i] != 0:
                Freq_Matrix[i][j] = Freq_Matrix[i][j]/TF[i]
            else:
                Freq_Matrix[i][j] = 0


    # store TF-IDF value of each cell in Matrix
    for i in range(len(corpus)):
        for j in range(len(vocabulary)):
            if(Freq_Matrix[i][j] > 0):
                Freq_Matrix[i][j] = IDF[j] * Freq_Matrix[i][j]

    Freq_Matrix = normalize(Freq_Matrix,norm = "l2")

    return Freq_Matrix
```

In [45]:

```
def Dense_matrix(Matrix,k):
     doc = dict()
     for i in range(len(Matrix[k])):
         doc[(k,i)] = Matrix[k][i]

     # Sort the dictionary based on value(tf-idf value) in non-increasing order
     result = dict(sorted(doc.items(),key = lambda item:item[0],reverse = True ))

     # print only non-zero values and corresponding index in matrix
     for key,value in result.items():
         if(value > 0):
             print("{} : {}".format(key,value))
```

In [50]:

```
## SkLearn# Collection of string documents

corpus = [
     'this is the first document',
     'this document is the second document',
     'and this is the third one',
     'is this the first document',
]
vocab,idf = fit(corpus)
s_matrix = transform(corpus,vocab,idf)
print("List of all Unique words in Corpus : ",list(vocab.keys()),"\n")

print("List of IDF values of each word : ",idf,"\n")

print("Shape of Matrix : ",s_matrix.shape,"\n")
print(s_matrix[3],"\n")
print(Dense_matrix(s_matrix,3),"\n")
```

```
List of all Unique words in Corpus :  ['and', 'document', 'first', 'is', 'one', 'secon
d', 'the', 'third', 'this']

List of IDF values of each word :  [1.916290731874155, 1.2231435513142097, 1.51082562376
59907, 1.0, 1.916290731874155, 1.916290731874155, 1.0, 1.916290731874155, 1.0]

Shape of Matrix :  (4, 9)

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

(3, 8) : 0.3840852409148149
(3, 6) : 0.3840852409148149
(3, 3) : 0.3840852409148149
(3, 2) : 0.580285823684436
(3, 1) : 0.4697913855799205
None
```

SKlearn Implementation

In [51]:

```python
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
vectorizer.fit(corpus)
skl_output = vectorizer.transform(corpus)
# sklearn feature names, they are sorted in alphabetic order by default.

print(vectorizer.get_feature_names())
# Here we will print the sklearn tfidf vectorizer idf values after applying the fit met
# After using the fit function on the corpus the vocab has 9 words in it, and each has

print(vectorizer.idf_)
# shape of sklearn tfidf vectorizer output after applying transform method.

skl_output.shape
# sklearn tfidf values for first line of the above corpus.
# Here the output is a sparse matrix
print(skl_output[3])
```

```
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
[1.91629073 1.22314355 1.51082562 1.         1.91629073 1.91629073
 1.         1.91629073 1.        ]
  (0, 8)	0.38408524091481483
  (0, 6)	0.38408524091481483
  (0, 3)	0.38408524091481483
  (0, 2)	0.5802858236844359
  (0, 1)	0.46979138557992045
```

Task - II

In [48]:

```python
def fit50(corpus):
    unique_words = set() # This SET will contain all unique words of corpus
    idf = [] # list containing top 50 idf values
    #If corpus is list ,Add words to set where len is greter than 1
    if(isinstance(corpus,list)):
        for doc in corpus:
            for word in doc.split(" "):
                if len(word)<2 :
                    continue
                unique_words.add(word)
        # Sort it, With list then enumate to create dictionary to give value to each un
        unique_words = sorted(list(unique_words))
        vocabulary = {j:i for i,j in enumerate(unique_words)}

        #count occurance of each word in corpus and sort them in increasing order
        count_words = dict()
        if(isinstance(corpus,list)):
            for word in vocabulary:
                count = 0
                for doc in corpus:
                    if word in doc.split():
                        count+=1
                count_words[word] = count
            count_words = sorted(count_words.items(),key = lambda item:item[1],reverse

        # create dict with top 50 less freq words
        new_vocabulary_count = dict()
        count = 0
        for i,j in count_words:
            if count < 50:
                new_vocabulary_count.update({i:j})
                count+=1
            else:
                break

        unique_words = sorted(list(new_vocabulary_count.keys()))
        new_vocabulary = {j:i for i,j in enumerate(unique_words)}

        # calculate idf values for words in new_vocab and store in idf list
        for word in new_vocabulary.keys():
            count = 0
            for doc in corpus:
                if word in doc.split(" "):
                    count+=1
            k = math.log((len(corpus)+1)/(count+1))+1
            idf.append(k)

        return new_vocabulary,idf
```

In [ ]:

In [49]:

```
# Below is the code to load the cleaned_strings pickle file provided
# Here corpus is of list type

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

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

vocab,idf = fit50(corpus)
s_matrix = transform(corpus,vocab,idf)
print("List of top 50 idf value Unique words in Corpus : ",list(vocab.keys()),"\n")
print("List of IDF values of each word : ",idf,"\n")

print("Shape of Matrix : ",s_matrix.shape,"\n")
print(corpus[0],"\n")
print(s_matrix[0],"\n")
print(Dense_matrix(s_matrix,0),"\n")
```

```
Number of documents in corpus =  746
List of top 50 idf value Unique words in Corpus :  ['aailiyah', 'abandoned', 'abroad',
'abstruse', 'academy', 'accents', 'accessible', 'acclaimed', 'accolades', 'accurate', 'a
ccurately', 'achille', 'ackerman', 'actions', 'adams', 'add', 'added', 'admins', 'admira
tion', 'admitted', 'adrift', 'adventure', 'aesthetically', 'affected', 'affleck', 'after
noon', 'aged', 'ages', 'agree', 'agreed', 'aimless', 'aired', 'akasha', 'akin', 'alert',
'alike', 'allison', 'allow', 'allowing', 'alongside', 'amateurish', 'amaze', 'amazed',
'amazingly', 'amusing', 'amust', 'anatomist', 'angel', 'angela', 'angelina']

List of IDF values of each word :  [6.922918004572872, 6.922918004572872, 6.922918004572
872, 6.922918004572872, 6.922918004572872, 6.922918004572872, 6.922918004572872, 6.92291
8004572872, 6.922918004572872, 6.922918004572872, 6.922918004572872, 6.922918004572872,
6.922918004572872, 6.922918004572872, 6.922918004572872, 6.922918004572872, 6.9229180045
72872, 6.922918004572872, 6.922918004572872, 6.922918004572872, 6.922918004572872, 6.922
918004572872, 6.922918004572872, 6.922918004572872, 6.922918004572872, 6.92291800457287
2, 6.922918004572872, 6.922918004572872, 6.922918004572872, 6.922918004572872, 6.9229180
04572872, 6.922918004572872, 6.922918004572872, 6.922918004572872, 6.922918004572872, 6.
922918004572872, 6.922918004572872, 6.922918004572872, 6.922918004572872, 6.922918004572
872, 6.922918004572872, 6.922918004572872, 6.922918004572872, 6.922918004572872, 6.92291
8004572872, 6.922918004572872, 6.922918004572872, 6.922918004572872, 6.922918004572872,
6.922918004572872]

Shape of Matrix :  (746, 50)

slow moving aimless movie distressed drifting young man

[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.]

(0, 30) : 1.0
None
```

In [ ]: