In [41]:

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

In [42]:

```
def idf_list(vocab,dataset):
    print("words and idf values : ")
    idf_values=[]
    for i in vocab:
        count=0
        for sentence in dataset:#occorences of word in corpus text
            if i in sentence:
                count+=1
        idf=1+math.log((1+ len(dataset))/(1+ count))
        idf_values.append(idf)
        print("{0} : {1}".format(i,idf))  #print words and idf values
    return idf_values
def fit(dataset): #task1
    unique=set()
    if isinstance(dataset,(list,)): #check if dataset is of list type
        for sentence in dataset:
            for word in sentence.split(" "):
                if(len(word)<2): #remove words like , ' a
                      continue
                unique.add(word) #set of unique words
        unique_words=sorted(list(unique))
        idf_values=idf_list(unique_words,dataset) #function to print vocab words and idf
values
        vocab={j : i for i,j in enumerate(unique_words) }
        return vocab , idf_values
    else:
        print("pass list")

vocab, idf_values=fit(dataset)
print("feature names and column numbers")
print(vocab)
print(idf_values)
```

```
words and idf values :
and : 1.916290731874155
document : 1.2231435513142097
first : 1.5108256237659907
is : 1.0
one : 1.916290731874155
second : 1.916290731874155
the : 1.0
third : 1.916290731874155
this : 1.0
feature names and column numbers
{'and': 0, 'document': 1, 'first': 2, 'is': 3, 'one': 4, 'second': 5, 'the': 6, 'third':
7, 'this': 8}
[1.916290731874155, 1.2231435513142097, 1.5108256237659907, 1.0, 1.916290731874155, 1.916
290731874155, 1.0, 1.916290731874155, 1.0]
```

In [33]:

```
def transform(dataset,vocab,idf_values):
    rows=[]
    columns=[]
    values=[]
    if isinstance(dataset,(list,)):
        for id,doc in enumerate((dataset)):

            word_fre=dict(Counter(doc.split())) #making dictionary
            for word,freq in word_fre.items():
                col_index=vocab.get(word,-1) #if word not present return -1 or else colu
mn number
                if(col_index!=-1):
                      rows.append(id) #row no. of selected feature
                      columns.append(col_index) #column no.
                      tf=freq/float(len(doc.split(" "))) #no. of times word occured in d
ocument
                      a=idf_values[col_index]
                      x=tf*a
                      values.append(x) #idf*tf value of feature word
            return normalize(csr_matrix( ((values),(rows,columns)), shape=(len(dataset),
len(vocab))),norm='l2' ) #creating sparse matrix and normalizing

cus_output=transform(dataset,vocab,idf_values)
print(cus_output[0]) #sparse matrix
print(cus_output[0].toarray())#dense matrix of first line
print(cus_output.shape) #dimension (rows,col)
```

```
  (0, 1) 0.4697913855799205
  (0, 2) 0.580285823684436
  (0, 3) 0.3840852409148149
  (0, 6) 0.3840852409148149
  (0, 8) 0.3840852409148149
[[0.         0.46979139 0.58028582 0.38408524 0.         0.
  0.38408524 0.         0.38408524]]
(4, 9)
```

this output matches perfectly with that of sklearn vectorizer

---

In [43]:

```
# TASK 2
#https://www.w3resource.com/python-exercises/dictionary/python-data-type-dictionary-exerc
ise-1.php
def idf_list2(v,corpus): #for calculating top idf values and their words
    sort_idf=[]
    d={}
    for i in v:
        count=0
        for sentence in corpus:
            if i in sentence.split(" "):
                count+=1
        idf=1+math.log((1+ len(corpus))/(1+ count))
        sort_idf.append(idf)
    d=dict(zip(v,sort_idf))  #dict of unique_words and idf values
    sorted_d = dict(sorted(d.items(), key=operator.itemgetter(1),reverse=True))#sorting
in descending idf values    z=0
    sorted_vocab=[]
    sorted_idf=[]
    print("top 50 idf keys and their values :")
    z=0
    for k,v in sorted_d.items(): #printing top 50 idf values and words
        sorted_vocab.append(k)
        sorted_idf.append(v)
        print("{0} : {1}".format(k,v))
```

```
        z+=1
        if(z==50):
            return sorted_vocab , sorted_idf  #returning top idf value feature names
            break


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

def fit(corpus):
    unique=set()
    if isinstance(corpus,(list,)):
        for sentence in corpus:
            for word in sentence.split(" "):
                if(len(word)<2):
                      continue

                unique.add(word) #unique set
        unique_words=sorted(list(unique))
        final_words, sorted_idf=idf_list2(unique_words,corpus)#function returns top 50 fe
atures and idf values

        v=dict(zip(final_words,list(range(50)))) #dict of unique words and their column
index
        return v ,sorted_idf
    else:
        print("pass list")

tt,sorted_idf=fit(corpus) #return unique_words and idf values
```

```
top 50 idf keys and their values :
aailiyah : 6.922918004572872
abandoned : 6.922918004572872
abroad : 6.922918004572872
abstruse : 6.922918004572872
academy : 6.922918004572872
accents : 6.922918004572872
accessible : 6.922918004572872
acclaimed : 6.922918004572872
accolades : 6.922918004572872
accurate : 6.922918004572872
accurately : 6.922918004572872
achille : 6.922918004572872
ackerman : 6.922918004572872
actions : 6.922918004572872
adams : 6.922918004572872
add : 6.922918004572872
added : 6.922918004572872
admins : 6.922918004572872
admiration : 6.922918004572872
admitted : 6.922918004572872
adrift : 6.922918004572872
adventure : 6.922918004572872
aesthetically : 6.922918004572872
affected : 6.922918004572872
affleck : 6.922918004572872
afternoon : 6.922918004572872
aged : 6.922918004572872
ages : 6.922918004572872
agree : 6.922918004572872
agreed : 6.922918004572872
aimless : 6.922918004572872
aired : 6.922918004572872
akasha : 6.922918004572872
akin : 6.922918004572872
alert : 6.922918004572872
alike : 6.922918004572872
allison : 6.922918004572872
allow : 6.922918004572872
allowing : 6.922918004572872
alongside : 6.922918004572872
```

```
amateurish : 6.922918004572872
amaze : 6.922918004572872
amazed : 6.922918004572872
amazingly : 6.922918004572872
amusing : 6.922918004572872
amust : 6.922918004572872
anatomist : 6.922918004572872
angel : 6.922918004572872
angela : 6.922918004572872
angelina : 6.922918004572872
```

In [53]:

```
def transform2(corpus,tt,sorted_idf):
    rows2=[]
    columns2=[]
    values2=[]
    for id,doc in enumerate((corpus)):
        word_fre=dict(Counter(doc.split()))
        for word,freq in word_fre.items():
            if(len(word)< 2):
                continue
            col_index=tt.get(word,-1) #if word in top feature names return col_index or
else -1
            if(col_index!=-1):

                  rows2.append(id) #row no. of corpus text containing word
                  columns2.append(col_index) #column number of feature name
                  tf=freq/float(len(doc.split(" ")))
                  a=sorted_idf[col_index] #a is idf value of fit function
                  b=a*tf
                  values2.append(b)

#normalizing  plus sparse matrix
    return normalize(csr_matrix( ((values2),(rows2,columns2)), shape=(len(corpus),len(tt
))),norm='l2' )

cus_output=transform2(corpus,tt,sorted_idf)
print(cus_output)
print(cus_output[0].toarray()) #sparse matrix representation
print("no of columns in dense matrix :",len(i))
```

```
  (0, 30) 1.0
  (68, 24) 1.0
  (72, 29) 1.0
  (74, 31) 1.0
  (119, 33) 1.0
  (135, 3) 0.37796447300922725
  (135, 10) 0.37796447300922725
  (135, 18) 0.37796447300922725
  (135, 20) 0.37796447300922725
  (135, 36) 0.37796447300922725
  (135, 40) 0.37796447300922725
  (135, 41) 0.37796447300922725
  (176, 49) 1.0
  (181, 13) 1.0
  (192, 21) 1.0
  (193, 23) 1.0
  (216, 2) 1.0
  (222, 47) 1.0
  (225, 19) 1.0
  (227, 17) 1.0
  (241, 44) 1.0
  (270, 1) 1.0
  (290, 25) 1.0
  (333, 26) 1.0
  (334, 15) 1.0
  (341, 43) 1.0
  (344, 42) 1.0
  (348, 8) 1.0
  (377, 37) 1.0
  (409, 5) 1.0
```

```
  (430, 39) 1.0
  (457, 45) 1.0
  (461, 4) 1.0
  (465, 38) 1.0
  (475, 35) 1.0
  (493, 6) 1.0
  (500, 48) 1.0
  (548, 0) 0.7071067811865475
  (548, 32) 0.7071067811865475
  (608, 14) 1.0
  (612, 11) 1.0
  (620, 46) 1.0
  (632, 7) 1.0
  (644, 12) 0.7071067811865475
  (644, 27) 0.7071067811865475
  (664, 28) 1.0
  (667, 22) 1.0
  (691, 34) 1.0
  (697, 9) 1.0
  (722, 16) 1.0
[[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.]]
no of columns in dense matrix : 50
```

In [0]:

```
```

In [0]:

```
```