TFIDF Implementation without using scikit-learn

In [1]:
```
# Taking a simple corpus example for this implementation without any capital letters and puntuations as
 we need to
# compare our results with scikit-learn's implementation and scikit-lean deals with such strings differ
ently.
corpus = [
     'this is the first document',
     'this document is the second document',
     'and this is the third one',
     'is this the first document',
        ]
```

Your custom implementation

In [2]:
```
# Importing libraries
from collections import Counter
from scipy.sparse import csr_matrix
import math
from sklearn.preprocessing import normalize
```

In [3]:
```
# This function returns all the unique words from the corpus in an ascending order.
def fit(data):

    vocab={} # Empty dictionary which would contain all the unique words.
    temp=[]

    for sen in data: # To iterate through each document in the corpus.
        sen=sen.split()
        #print(sen)

        for word in sen: # To through each word in a document.
            #print(word)
            if word not in temp:
                temp.append(word) # Adding only the unique words in the list temp.

    temp=sorted(temp) # Sorting the words alphabetically.

    for i in range(len(temp)):
        vocab[temp[i]]=i # Adding indices to the sorted unique words.

    return vocab
```

In [4]:
```
vocab = fit(corpus)
print(vocab.keys())
```

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

In [5]:
```
# Calculating the IDF values for all the unique words generated from the corpus.
def calc_idf(data,bag):

    idf=[] # Empty list which would contain all the IDF values of the unique words respectively.
    N=len(data) # Getting the numerator value for the IDF formula which is total number of documents in
 a corpus.

    for word in bag.keys(): # Iterating through the each word from the bag of words.
        deno=0 # Initializing denominator 0 for each word in the bag of words.
        #print(word)

        for sen in data: # Iterating through the each document in the corpus.
            sen=sen.split()
            #print(sen)
            if word in sen: # Checking if the word is available in the particular document.
                deno+=1

        ln=(N+1)/(deno+1)
        ln=1 + math.log(ln) # Calculating IDF.
        #print(idf)
        idf.append(ln) # Adding each of the IDF value generated in the idf list.

    return idf
```

In [6]:
```
idf = calc_idf(corpus,vocab)
print(idf)
```

```
[1.9162907318741455, 1.2231435513142097, 1.5108256237659907, 1.0, 1.9162907318741455, 1.916290731874155
5, 1.0, 1.9162907318741455, 1.0]
```

In [7]:
```
# Transform function which would return the sparse matrix of the TF-IDF values of the documents in a co
rpus.
def calc_transform(data,vocab):

    rows=[] # Empty list which would contain row values for the sparse matrix.
    columns=[] # Empty list which would contain column values for the sparse matrix.
    values=[] # Empty list which would contain TF-IDF values for the sparse matrix.

    for sen in range(len(data)): # Iterating through each document in the corpus.
        temp=dict(Counter(data[sen].split())) # Converting each document into the dictionary with keys
 as word and values as
                                              # occurence of each word in that document.
        #print(temp)

        for key,value in temp.items(): # Iterating through keys and values in the dictionary created ab
ove.
            col=vocab.get(key) # Retrieving index of the document word from the bag of words dictionar
y.
            #print(sen,col,value)
            deno=len(data[sen].split()) # Retrieving denominator for TF formula which total number word
s in a document.
            #print(deno)
            tf=value/deno # Calculating TF.
            #print(tf)
            #print(idf[col])
            tfidf=tf*idf[col] # Calculating TF-IDF value by retrieving respective IDF values calculated
 above.
            #print('tfidf',tfidf)
            rows.append(sen) # Adding respective row values which are required for sparse matrix creati
on.
            columns.append(col) # Adding respective columns values which are required for sparse matrix
 creation.
            values.append(tfidf) # Adding respective TF-IDF values which are required for sparse matrix
 creation.

    #print(rows,columns,values)
    mat=csr_matrix((values,(rows,columns)),shape=(len(data),len(vocab))) #Getting sparse matrix from pa
rameters retrieved above.
    #print(mat[0])
    mat=normalize(mat) # Applying L2 Normalization on the sparse matrix.
    #print(mat[0])

    return mat
```

In [9]:
```
transformed = calc_transform(corpus,vocab)
print('TFIDF:',transformed[0].toarray())
print('IDF:',idf)
```

```
TFIDF: [[0.         0.46979139 0.58028582 0.38408524 0.         0.
  0.38408524 0.         0.38408524]]
IDF: [1.9162907318741455, 1.2231435513142097, 1.5108256237659907, 1.0, 1.9162907318741455, 1.9162907318
74155, 1.0, 1.9162907318741455, 1.0]
```

Using scikit-learn to compare my results.

In [10]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
vectorizer.fit(corpus)
skl_output = vectorizer.transform(corpus)
```

In [11]:
```
# sklearn feature names, they are sorted in alphabetic order by default.
print(vectorizer.get_feature_names())
```

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

In [12]:
```
# Here we will print the sklearn tfidf vectorizer idf values after applying the fit method
# After using the fit function on the corpus the vocab has 9 words in it, and each has its idf value.

print(vectorizer.idf_)
```

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

In [13]:
```
from sklearn.feature_extraction.text import CountVectorizer

vec = CountVectorizer(analyzer='word')

vec.fit(corpus)
feature_matrix_2 = vec.transform(corpus)
print(feature_matrix_2.toarray())
```

```
[[0 1 1 1 0 0 1 0 1]
 [0 2 0 1 0 1 1 0 1]
 [1 0 0 1 1 0 1 1 1]
 [0 1 1 1 0 0 1 0 1]]
```

In [14]:
```
# shape of sklearn tfidf vectorizer output after applying transform method.
skl_output.shape
```

Out[14]:
```
(4, 9)
```

In [15]:
```
# sklearn tfidf values for first line of the above corpus.
# Here the output is a sparse matrix
print(skl_output[0])
```

```
  (0, 8)        0.38408524091481483
  (0, 6)        0.38408524091481483
  (0, 3)        0.38408524091481483
  (0, 2)        0.5802858236844359
  (0, 1)        0.4697913857992045
```

In [16]:
```
# To understand the output better, here we are converting the sparse output matrix to dense matrix and
 printing it.
# Notice that this output is normalized using L2 normalization. sklearn does this by default.
print(skl_output[0].toarray())
```

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

Our implementation values and scikit-learn's values both are same.