TFIDF

1) SKLearn Implementation

In [1]:

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

In [2]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
vectorizer.fit(corpus)
skl_output = vectorizer.transform(corpus)
```

In [3]:

```
# 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 [4]:

```
# After using the fit function on the corpus the vocab has 9 words in it, and each has its idf val
ue.

print(vectorizer.idf_)
```

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

In [5]:

```
# shape of sklearn tfidf vectorizer output after applying transform method.

skl_output.shape
```

Out[5]:

```
(4, 9)
```

In [6]:

```
# 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)	0.46979138557992045
```

In [7]:

```
# sklearn tfidf values for first line of the above corpus.
# 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]]
```

2) With out SKLearn,TFIDF from scatch

In [8]:

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

In [9]:

```
## Importing libraries ##
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 [10]:

```
## Fit method ##
import pandas as pd
def fit(corpus):
    unique_words = set()
    if isinstance(corpus, (list,)):
        for row in corpus:
            for word in row.split(" "):
                if len(word) < 2:
                    continue
                unique_words.add(word)
        unique_words = sorted(list(unique_words))
        vocab = {i:j for i,j in enumerate(unique_words)}
        return vocab
    else:
        print("you need to pass list of sentence")

vocab=fit(corpus)
print(vocab)
```

```
{0: 'and', 1: 'document', 2: 'first', 3: 'is', 4: 'one', 5: 'second', 6: 'the', 7: 'third', 8: 'th
is'}
```

In [12]:

```
words=[]
for row in corpus:
  t=[]
  for word in row.split(" "):
    t.append(word)
  words.append(t)
## Transform method ##
temp = []
tf=[]
for row in words:
  temp=[0]*len(vocab)
  for word in row:
```

```
    for w in vocab:
      if vocab[w]==word:
        temp[w]=temp[w]+1
  tf.append(temp)
  temp=[0]*len(vocab)
print(tf)
```

```
[[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 [13]:

```
## Checking the feature names sorted in alphebatical order from sklearn ##
a=vocab.values()
b=sorted(a)
print(b)
```

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

In [14]:

```
## Calculating Tf ##
for i in range(len(tf)):
  for j in range(len(tf[i])):
    tf[i][j]=(tf[i][j])/len(words[i])
print(tf)
```

```
[[0.0, 0.2, 0.2, 0.2, 0.0, 0.0, 0.2, 0.0, 0.2], [0.0, 0.3333333333333333, 0.0, 0.16666666666666666, 0.0, 0.16666666666666666, 0.16666666666666666, 0.0, 0.16666666666666666], [0.16666666666666666, 0.0, 0.0, 0.16666666666666666, 0.16666666666666666, 0.0, 0.16666666666666666, 0.16666666666666666], [0.0, 0.2, 0.2, 0.2, 0.0, 0.0, 0.2, 0.0, 0.2]]
```

In [15]:

```
## Number of occurances ##
Oc=[]
for i in vocab:
  t=0
  for row in corpus:
    for word in row.split(" "):
      if(vocab[i]==word):
        t=t+1;
        break;
  Oc.append(t)
  print(vocab[i] + " - " + str(Oc[i]))
```

```
and - 1
document - 3
first - 2
is - 4
one - 1
second - 1
the - 4
third - 1
this - 4
```

In [16]:

```
## Calculating IDF ##
import math
idf=[]
N=len(corpus)
for i in vocab:
  t=0;
  t = 1 + (math.log((N+1)/(Oc[i]+1)))
  idf.append(t)
print(idf)
```

```
[1.916290731874155, 1.2231435513142097, 1.5108256237659907, 1.0, 1.916290731874155,
```

```
1.9162907318741551, 1.0, 1.9162907318741551, 1.0]
```

In [17]:

```
## Calculating TFIDF ##
for i in range(len(tf)):
  for j in range(len(tf[i])):
    tf[i][j]=(tf[i][j])*(idf[j])
print(tf)
```

```
[[0.0, 0.24462871026284194, 0.3021651247531982, 0.2, 0.0, 0.0, 0.2, 0.0, 0.2], [0.0,
0.40771451710473655, 0.0, 0.16666666666666666, 0.0, 0.3193817886456925, 0.16666666666666666, 0.0,
0.16666666666666666], [0.3193817886456925, 0.0, 0.0, 0.16666666666666666, 0.3193817886456925, 0.0,
0.16666666666666666, 0.3193817886456925, 0.16666666666666666], [0.0, 0.24462871026284194,
0.3021651247531982, 0.2, 0.0, 0.0, 0.2, 0.0, 0.2]]
```

In [18]:

```
## L2 Normalization ##
from sklearn.preprocessing import normalize
tf_normalized= normalize(tf, norm='l2',axis=1, copy=True, return_norm=False)
print(tf_normalized)
```

```
[[0.         0.46979139 0.58028582 0.38408524 0.         0.
  0.38408524 0.         0.38408524]
 [0.         0.6876236  0.         0.28108867 0.         0.53864762
  0.28108867 0.         0.28108867]
 [0.51184851 0.         0.         0.26710379 0.51184851 0.
  0.26710379 0.51184851 0.26710379]
 [0.         0.46979139 0.58028582 0.38408524 0.         0.
  0.38408524 0.         0.38408524]]
```

In [19]:

```
## Sparse matrix ##
from scipy.sparse import csr_matrix
from scipy import sparse
import numpy as np
output = sparse.csr_matrix(tf_normalized)
print(output[0])
```

```
  (0, 1) 0.4697913855799205
  (0, 2) 0.580285823684436
  (0, 3) 0.3840852409148149
  (0, 6) 0.3840852409148149
  (0, 8) 0.3840852409148149
```

In [20]:

```
## Converting sparse matrix to Dense matrix ##
print(output[0].toarray())
```

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

In [ ]:

```
```