Assignment

Task-1

Corpus

In [1]:

```
## 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' ]
```

SkLearn Implementation

In [2]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
vectorizer.fit(corpus)
print(vectorizer.get_feature_names())   # sklearn feature names, they are sorted in alphabetic order by de
```

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

In [3]:

```
# 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 [4]:

```
# shape of sklearn tfidf vectorizer output after applying transform method.
skl_output = vectorizer.transform(corpus)
skl_output.shape
```

Out[4]:

```
(4, 9)
```

In [5]:

```
# 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 [6]:

```
# 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 pri
# 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]]
```

My custom implementation

In [7]:

```
# Required imports for my custom Implementation
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 as np
```

In [8]:

```
# Fit fucntion to define the features (unique words)
def fit(dataset):
    unique_words = set() # at first we will initialize an empty set
    # check if its list type or not
    if type(dataset) == list:
        for review in dataset: # for each review in the dataset
            for word in review.split(" "): # for each word in the review. #split method converts each stri
                if len(word) >= 2:
                    unique_words.add(word)
        return sorted(list(unique_words))
    else:
        print("Invalid Input. Please give corpus as a list")
My_Custom_features = fit(corpus)
print(My_Custom_features)
```

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

In [9]:

```
# function to calculate IDF vale of any word in given corpus
def IDF(t, dataset):
    n = 0               #Number_of_documents_having_t = 0
    N = len(dataset)    # total number of Docs in corpes
    for review in dataset: # for each review in the dataset
        for word in review.split(" "): # for each word in the review. #split method converts each string i
            if word ==t:
                n +=1 ;
                break   # goes to nextsentence at first occurance of the t in a document
    return 1+math.log((1+N)/(1+n))

idf = [IDF(word, corpus) for word in My_Custom_features] #list of IDF value in same order as vocab
idf
```

Out[9]:

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

In [10]:

```
# final custom tf-idf Vectorizer
def transform(Corpus, Dimensions):
    loc_r, loc_c, values = [], [], []

    if type(Corpus) == list:
        r = 0    # to store row index in sparse matric
        for review in Corpus:       # sets row
            word_freq = dict(Counter(review.split()))  # counts occurance of a word in a doc and stores as
            c = 0    # to store col index in sparse matric
            for word in Dimensions:       # sets column
                if word in review.split(): # checks if the word is in present row
                    loc_r.append(r) ; loc_c.append(c)
                    values.append(((word_freq[word])*IDF(word, corpus))/len(review.split()))
                c +=1
            r +=1
        return normalize(csr_matrix((values, (loc_r, loc_c)), shape=(len(Corpus),len(Dimensions))), "l2")
    else:
        print("you need to pass list of strings")

My_Custom_Output = transform(corpus,My_Custom_features)
My_Custom_Output.shape
print(My_Custom_Output[0])
```

```
  (0, 1)        0.4697913855799205
  (0, 2)        0.5802858236844359
  (0, 3)        0.3840852409148149
  (0, 6)        0.3840852409148149
  (0, 8)        0.3840852409148149
```

In [11]:

```
print(My_Custom_Output[0].toarray())
```

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

Task-2

In [5]:
```
# Required imports for my custom Implementation
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 as np
```

In [6]:
```
import pickle
with open('/content/drive/My Drive/Colab Notebooks/3_CountVectorizer/cleaned_strings', 'rb') as f:
    corpus_2 = pickle.load(f)

# printing the length of the corpus loaded
print("Number of documents in corpus = ",len(corpus_2))
```

```
Number of documents in corpus =  746
```

In [7]:
```
# fit function to return vocab as unique values
def fit(Corpus):
    unique_words = set() # at first we will initialize an empty set
    # check if its list type or not
    if type(Corpus) == list:
        for review in Corpus: # for each review in the Corpus
            for word in review.split(" "): # for each word in the review. #split method converts each stri
                if len(word) >= 2:
                    unique_words.add(word)
        return sorted(list(unique_words))
    else:
        print("Invalid Input. Please give corpus as a list")
vocab = fit(corpus_2)
print(len(vocab))
```

```
2886
```

In [8]:
```
def IDF(t, Corpus):
    n = 0               #Number_of_documents_having_t = 0
    N = len(Corpus)    # total number of Docs in corpes
    for doc in Corpus: # for each doc in the Corpus
        for word in doc.split(" "): # for each word in the doc. #split method converts each string into Li
            if word ==t:
                n +=1 ;
                break   # goes to nextsentence at first occurance of the t in a document
    return 1+math.log((1+N)/(1+n))

list_idf = [IDF(word, corpus_2) for word in vocab] #list of IDF value in same order as vocab
print(len(list_idf))
print(list_idf[:5])
```

```
2886
[6.922918004572872, 6.922918004572872, 6.229770824012927, 6.922918004572872, 5.313480092138771 5]
```

In [22]:
```
# ordered idf values for alphabetical words in the vocab
idf_vocab = tuple(zip(vocab, list_idf))    # tuple of word along with its idf value as (word, idf)
list_idf.sort(reverse = True)              # sorting idf values in descending order
Rank_list = []
for idf in list_idf:
    for entry in idf_vocab:
        if entry[0] in Rank_list:
            continue
        if entry[1] == idf:
            Rank_list.append(entry[0]) ; break
Dimensions = Rank_list[:50]          # Top 50 idf words as dimensions
print(Dimensions)
```

```
['aailiyah', 'abandoned', 'ability', 'abroad', 'absolutely', 'abstruse', 'abysmal', 'academy', 'accent
s', 'accessible', 'acclaimed', 'accolades', 'accurate', 'accurately', 'accused', 'achievement', 'achill
e', 'ackerman', 'act', 'acted', 'acting', 'action', 'actions', 'actor', 'actors', 'actress', 'actresse
s', 'actually', 'adams', 'adaptation', 'add', 'added', 'addition', 'admins', 'admiration', 'admitted',
'adorable', 'adrift', 'adventure', 'advise', 'aerial', 'aesthetically', 'affected', 'affleck', 'afraid',
'africa', 'afternoon', 'age', 'aged', 'ages']
```

In [25]:

```
# final custom tf-idf Vectorizer
def transform(Corpus, Dimensions):
    loc_r, loc_c, values = [], [], []
    if type(Corpus) == list:
        r = 0
        for doc in Corpus: # for each document in the Corpus
            word_freq = dict(Counter(doc.split())) ; c = 0
            for word in Dimensions:
                if word in doc.split():

                    loc_r.append(r) ; loc_c.append(c)
                    values.append(((word_freq[word])*IDF(word, Corpus))/len(doc.split()))
                c +=1
            r +=1
        normalized_sparse_matrix = normalize(csr_matrix((values, (loc_r, loc_c)), shape=(len(Corpus),len(
        return normalized_sparse_matrix
    else:
        print("you need to pass list of strings")
Custom_TF_IDF = transform(corpus_2,Dimensions)
print(Custom_TF_IDF[10].toarray())
Custom_TF_IDF.shape
```

```
[[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. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0.]]
```

Out[25]:

```
(746, 50)
```