AAIC assignment 3 (TFID)

```
import warnings
warnings.filterwarnings("ignore")
from collections import Counter
from tqdm import tqdm
from scipy.sparse import csr_matrix
import math as m
import operator
from sklearn.preprocessing import normalize
import numpy


from tqdm import tqdm # tqdm is a library that helps us to visualize the runtime of
#https://tqdm.github.io/

# it accepts only list of sentances
def fit(dataset):
    unique_words = set() # at first we will initialize an empty set
    # 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.
                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)}



        return vocab
    else:
        print("you need to pass list of sentance")


corpus = [
     'this is the first document',
     'this document is the second document',
     'and this is the third one',
     'is this the first document',
]
vocab = fit(corpus)
# print(unique_words)
print(vocab)
```

✓ 0s completed at 10:01 AM

```
    if isinstance(dataset, (list,)):
        for idx, row in enumerate((dataset)): # for each document in the dataset
            # it will return a dict type object where key is the word and values is
            word_freq = dict(Counter(row.split()))
            # for every unique word in the document

            for word, freq in word_freq.items():  # for each unique word in the rev
                if len(word) < 2:
                    continue
                # we will check if its there in the vocabulary that we build in fit
                # dict.get() function will return the values, if the key doesn't ex
                col_index = vocab.get(word, -1) # retreving the dimension number of
                # if the word exists
                if col_index !=-1:
                    # we are storing the index of the document
                    rows.append(idx)
                    # we are storing the dimensions of the word
                    columns.append(col_index)
                    # we are storing the tf*idf values of the each word in list nam
                    values.append(idf_dict[word]*freq/len(row.split()))
        return csr_matrix((values, (rows,columns)), shape=(len(dataset),len(vocab))
    else:
        print("you need to pass list of strings")


#this function returns the IDF values of words in corpus which we are passing throu
def idf(vocab):
  idf=[]
  idf_dict={}
  for key,value in vocab.items():
    c=0
    for j in corpus:
      if key in j:
        c+=1
    df=1+m.log((1+len(corpus))/(1+c))
    idf.append(df)
    idf_dict[key]=df
  return idf_dict



idf_dict=idf(vocab) #Dictionary of IDF values with vocab words as key and its IDF v
```

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

```
print(final_op[0].toarray())··#printing·TFIDF·values·of·1st·row·document·from·corpu
```

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

```
print(final_op[0])·#printing·TFIDF·values·of·1st·row·document·from·corpus·as·sparse
```

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

Task2

```
from tqdm import tqdm # tqdm is a library that helps us to visualize the runtime of
#https://tqdm.github.io/
```

```
# it accepts only list of sentances
def fit(dataset):
    unique_words = set() # initializing  empty set
    unique_words1=set()  # initializing  empty set
    # 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 metho
                if len(word) < 2:
                    continue
                unique_words.add(word)
        unique_words = sorted(list(unique_words))
```

```
    values = []
    if isinstance(dataset, (list,)):
        for idx, row in enumerate((dataset)): # for each document in the dataset
            # it will return a dict type object where key is the word and values is
            word_freq = dict(Counter(row.split()))
            # for every unique word in the document

            for word, freq in word_freq.items():  # for each unique word in the rev
                if len(word) < 2:
                    continue
                # we will check if its there in the vocabulary that we build in fit
                # dict.get() function will return the values, if the key doesn't ex
                col_index = vocab.get(word, -1) # retreving the dimension number of
                # if the word exists
                if col_index !=-1:
                    # we are storing the index of the document
                    rows.append(idx)
                    # we are storing the dimensions of the word
                    columns.append(col_index)
                    # we are storing the tf*idf values of the each word in a row
                    values.append(idf_dict[word]*freq/len(row.split()))
        return csr_matrix((values, (rows,columns)), shape=(len(dataset),len(vocab))
    else:
        print("you need to pass list of strings")


from google.colab import files
uploaded = files.upload()
import pickle
with open('cleaned_strings', 'rb') as f:
    corpus = pickle.load(f)
```

```
{'waster': 0, 'wasting': 1, 'wave': 2, 'waylaid': 3, 'wayne': 4, 'weaker': 5,
```

```
print(final_op)
```

```
       (19, 16)      0.5773502691896258
       (19, 32)      0.5773502691896258
       (19, 45)      0.5773502691896258
       (55, 5)       1.0
       (68, 19)      1.0
       (70, 9)       1.0
       (80, 14)      1.0
       (109, 49)     1.0
       (134, 4)      1.0
       (135, 6)      0.408248290463863
       (135, 8)      0.408248290463863
       (135, 20)     0.408248290463863
       (135, 26)     0.408248290463863
       (135, 27)     0.408248290463863
       (135, 38)     0.408248290463863
       (148, 3)      0.5773502691896257
       (148, 17)     0.5773502691896257
       (148, 42)     0.5773502691896257
       (155, 39)     1.0
       (191, 24)     1.0
       (222, 43)     1.0
       (251, 37)     1.0
```

```
  (0, 45)       0.5773502691896258
```

```
print(final_op[19].toarray())  #printing TFIDF values of 1st row document from corp
```

```
[[0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.57735027 0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.57735027 0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.57735027 0.         0.
  0.         0.        ]]
```

```
print(final_op[19]) #printing TFIDF values of 1st row document from corpus as spars
```

```
  (0, 16)       0.5773502691896258
  (0, 32)       0.5773502691896258
  (0, 45)       0.5773502691896258
```