Assignment

What does tf-idf mean?

Tf-idf stands for *term frequency-inverse document frequency*, and the tf-idf weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Variations of the tf-idf weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query.

One of the simplest ranking functions is computed by summing the tf-idf for each query term; many more sophisticated ranking functions are variants of this simple model.

Tf-idf can be successfully used for stop-words filtering in various subject fields including text summarization and classification.

How to Compute:

Typically, the tf-idf weight is composed by two terms: the first computes the normalized Term Frequency (TF), aka. the number of times a word appears in a document, divided by the total number of words in that document; the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

- **TF:** Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:
  $TF(t) = \frac{\text{Number of times term t appears in a document}}{\text{Total number of terms in the document}}$.
- **IDF:** Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:
  $IDF(t) = \log_e \frac{\text{Total number of documents}}{\text{Number of documents with term t in it}}$. for numerical stabiltiy we will be changing this formula little bit $IDF(t) = \log_e \frac{\text{Total number of documents}}{\text{Number of documents with term t in it}+1}$.

Example

Consider a document containing 100 words wherein the word cat appears 3 times. The term frequency (i.e., tf) for cat is then (3 / 100) = 0.03. Now, assume we have 10 million documents and the word cat appears in one thousand of these. Then, the inverse document frequency (i.e., idf) is calculated as log(10,000,000 / 1,000) = 4. Thus, the Tf-idf weight is the product of these quantities: 0.03 * 4 = 0.12. </p>

Task-1

1. Build a TFIDF Vectorizer & compare its results with Sklearn:

- As a part of this task you will be implementing TFIDF vectorizer on a collection of text documents.

- You should compare the results of your own implementation of TFIDF vectorizer with that of sklearns implemenation TFIDF vectorizer.

- Sklearn does few more tweaks in the implementation of its version of TFIDF vectorizer, so to replicate the exact results you would need to add following things to your custom implementation of tfidf vectorizer:
    1. Sklearn has its vocabulary generated from idf sroted in alphabetical order
    2. Sklearn formula of idf is different from the standard textbook formula. Here the constant **"1"** is added to the numerator and denominator of the idf as if an extra document was seen containing every term in the collection exactly once, which prevents zero divisions. $IDF(t) = 1 + \log_e \frac{1 + \text{Total number of documents in collection}}{1 + \text{Number of documents with term t in it}}$.
    3. Sklearn applies L2-normalization on its output matrix.
    4. The final output of sklearn tfidf vectorizer is a sparse matrix.

- Steps to approach this task:
    1. You would have to write both fit and transform methods for your custom implementation of tfidf vectorizer.
    2. Print out the alphabetically sorted voacb after you fit your data and check if its the same as that of the feature names from sklearn tfidf vectorizer.
    3. Print out the idf values from your implementation and check if its the same as that of sklearns tfidf vectorizer idf values.
    4. Once you get your voacb and idf values to be same as that of sklearns implementation of tfidf vectorizer, proceed to the below steps.
    5. Make sure the output of your implementation is a sparse matrix. Before generating the final output, you need to normalize your sparse matrix using L2 normalization. You can refer to this link https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.normalize.html
    6. After completing the above steps, print the output of your custom implementation and compare it with sklearns implementation of tfidf vectorizer.
    7. To check the output of a single document in your collection of documents, you can convert the sparse matrix related only to that document into dense matrix and print it.

Note-1: All the necessary outputs of sklearns tfidf vectorizer have been provided as reference in this notebook, you can compare your outputs as mentioned in the above steps, with these outputs.

Note-2: The output of your custom implementation and that of sklearns implementation would match only with the collection of document strings provided to you as reference in this notebook. It would not match for strings that contain capital letters or punctuations, etc, because sklearn version of tfidf vectorizer deals with such strings in a different way. To know further details about how sklearn tfidf vectorizer works with such string, you can always refer to its official documentation.

Note-3: During this task, it would be helpful for you to debug the code you write with print statements wherever necessary. But when you are finally submitting the assignment, make sure your code is readable and try not to print things which are not part of this task.

Corpus

In [109…

```
## 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)
skl_output = vectorizer.transform(corpus)
```

In [3]:

```
# sklearn feature names, they are sorted in alphabetic order by default.

print(vectorizer.get_feature_names_out())
```

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

In [4]:

```
# Here we will print the sklearn tfidf vectorizer idf values after applying the fit
# After using the fit function on the corpus the vocab has 9 words in it, and each h

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 t
# Notice that this output is normalized using L2 normalization. sklearn does this by

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

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

Your custom implementation

In [8]:

```
# Write your code here.
# Make sure its well documented and readble with appropriate comments.
# Compare your results with the above sklearn tfidf vectorizer
# You are not supposed to use any other library apart from the ones given below

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 [118…

```
# Reference from Assignment_3_Reference.ipynb; Author: AppliedAi
def tfidf_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. #split method
                if len(word) < 2:
                    continue
                unique_words.add(word)
        unique_words = sorted(list(unique_words)) # sorting the list of unique words
        vocab = {j:i for i,j in enumerate(unique_words)}

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

unique_words=list(tfidf_fit(corpus).keys()) # list containing unique words
vocab=tfidf_fit(corpus) # dictionary conataining unique words as key and column inde
print("Sklearn's feature",vectorizer.get_feature_names_out(),"\n\n")
print("*"*100)
print("\n\nCustom function features ",unique_words)
```

```
Sklearn's feature ['and' 'document' 'first' 'is' 'one' 'second' 'the' 'third' 'thi
s']


****************************************************************************************************
```

```
****************


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

In [12]:

```
def idf(dataset,unique_words): #unique words is a list of words:vocab obtained from
 idf_list=[]
 for term in unique_words:
    count=0
    for row in dataset:
      if term in row:
        count+=1
    idf_list.append(1+math.log((1+len(dataset))/(1+count)))
 return idf_list
```

In [119…

```
idf_list=idf(corpus,unique_words) # IDF frequency corresponding to each feature
print("Sklearn's IDF values",vectorizer.idf_,"\n\n")
print("*"*100)
print("\n\n Custom function idf values",idf_list)
```

```
Sklearn's IDF values [1.91629073 1.22314355 1.51082562 1.         1.91629073 1.91629
073
 1.         1.91629073 1.         ]


****************************************************************************************
****************


 Custom function idf values [1.916290731874155, 1.2231435513142097, 1.51082562376599
07, 1.0, 1.916290731874155, 1.916290731874155, 1.0, 1.916290731874155, 1.0]
```

In [19]:

```
def tfidf_transform(dataset,vocab,idf_list):

  index=[]
  feature= []
  values= []

  for idx,row in enumerate(dataset):
    words=dict(Counter(row.split()))
    for word,freq in words.items():
      if len(word)<2:
        continue
      feature_idx=vocab.get(word,-1) # get return -1 if word not in vocab
      if feature_idx!=-1: # if word is present in vocab then execute below code
        feature.append(feature_idx)
        index.append(idx)
        values.append(idf_list[feature_idx]*freq/len(row))

  sparse_matrix=csr_matrix((values,(index,feature)),shape=(len(dataset),len(vocab)))
  return normalize(sparse_matrix) # L2-normalization of sparse matrix
```

In [121…

```
print("sklearn tfidf values for first line of the above corpus\n",skl_output[0],"\n\
print("*"*100)
print("\n\n Custom function values of first document")
doc1=tfidf_transform(corpus,vocab,idf_list).toarray()[0]
for i in range(len(vocab)):
  if doc1[i]!=0:
    print("(0,{}) \t\t".format(i),doc1[i])
```

```
sklearn tfidf values for first line of the above corpus
   (0, 8)	0.38408524091481483
  (0, 6)	0.38408524091481483
  (0, 3)	0.38408524091481483
  (0, 2)	0.5802858236844359
  (0, 1)	0.46979138557992045


*************************************************************************************
****************


 Custom function values of first document
(0,1)		0.46979138557992045
(0,2)		0.5802858236844359
(0,3)		0.3840852409148149
(0,6)		0.3840852409148149
(0,8)		0.3840852409148149
```

In [126…

```
print("shape of sklearn tfidf vectorizer output after applying transform method\t\t"
print("Shape of matrix obtained by custome function is:\t\t\t\t\t",tfidf_transform(c
print("First document of sparse matrix:\n",tfidf_transform(corpus,vocab,idf_list).to
```

```
shape of sklearn tfidf vectorizer output after applying transform method
(4, 9)
Shape of matrix obtained by custome function is:
(4, 9)

First document of sparse matrix:
 [0.         0.46979139 0.58028582 0.38408524 0.         0.
 0.38408524 0.         0.38408524]
```

Task-2

2. Implement max features functionality:

- As a part of this task you have to modify your fit and transform functions so that your vocab will contain only 50 terms with top idf scores.

- This task is similar to your previous task, just that here your vocabulary is limited to only top 50 features names based on their idf values. Basically your output will have exactly 50 columns and the number of rows will depend on the number of documents you have in your corpus.

- Here you will be give a pickle file, with file name **cleaned_strings**. You would have to load the corpus from this file and use it as input to your tfidf vectorizer.

- Steps to approach this task:
    1. You would have to write both fit and transform methods for your custom implementation of tfidf vectorizer, just like in the previous task. Additionally, here you have to limit the number of features generated to 50 as described above.
    2. Now sort your vocab based in descending order of idf values and print out the words in the sorted voacb after you fit your data. Here you should be getting only 50 terms in your vocab. And make sure to print idf values for each term in your vocab.
    3. Make sure the output of your implementation is a sparse matrix. Before generating the final output, you need to normalize your sparse matrix using L2 normalization. You can

refer to this link https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.normalize.html

4. Now check the output of a single document in your collection of documents, you can convert the sparse matrix related only to that document into dense matrix and print it. And this dense matrix should contain 1 row and 50 columns.

In [113…

```
# Below is the code to Load the cleaned_strings pickle file provided
# Here corpus is of list type
print("Note: The pickle file is loaded from gdrive. Kindly change the path according
import pickle
with open('cleaned_strings', 'rb') as f: # importing file from gdrive.
    corpus2 = pickle.load(f)

# printing the length of the corpus loaded
print("Number of documents in corpus = ",len(corpus2))
```

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

In [ ]:

```
# Write your code here.
# Try not to hardcode any values.
# Make sure its well documented and readble with appropriate comments.
```

Run the code below for Task2

In [138…

```
# First tfidf_fit function is used to get list of unique words
unique_words2=list(tfidf_fit(corpus2).keys()) # fitting the data; list of all unique

# idf function is called to get idf values for the above unique words
idf_array=numpy.array(idf(corpus2,unique_words2))  # idf values corresponding to uni

#storing the index of Top 50 idf values
sorted_index=list(numpy.argsort(idf_array))[::-1][:50] # index of words with Top 50
print("Fucntions from the above are used and changes are made to vocab and idf compu
```

```
Fucntions from the above are used and changes are made to vocab and idf computation
```

In [129…

```
# Printing the vocab with Top 50 IDF values
print("Printing the vocab with Top 50 IDF values")
vocab2={} # creating a dictionary for vocab
idf_array=numpy.sort(idf_array,)[::-1][:50] # sorting the idf array by Top-50 values
print("index  word\t\tIDF Value")
for i,index_value in enumerate(sorted_index):
  vocab2[i]=unique_words2[index_value]
  print(i," ",unique_words2[index_value],"\t\t",idf_array[i])
```

```
Printing the vocab with Top 50 IDF values
index  word             IDF Value
0    zombiez             6.922918004572872
1    hugo                6.922918004572872
2    holds               6.922918004572872
3    hollander           6.922918004572872
4    homework            6.922918004572872
5    honestly            6.922918004572872
6    hopefully           6.922918004572872
7    hopeless            6.922918004572872
8    horrendously                6.922918004572872
9    horrid              6.922918004572872
```

```
10    horrified          6.922918004572872
11    hosting            6.922918004572872
12    houses             6.922918004572872
13    howdy              6.922918004572872
14    howell             6.922918004572872
15    humanity           6.922918004572872
16    hoffman            6.922918004572872
17    humans             6.922918004572872
18    hummh              6.922918004572872
19    hurt               6.922918004572872
20    hype               6.922918004572872
21    hypocrisy          6.922918004572872
22    idealogical                6.922918004572872
23    identified                 6.922918004572872
24    identifies                 6.922918004572872
25    idiotic            6.922918004572872
26    idyllic            6.922918004572872
27    imagine            6.922918004572872
28    imdb               6.922918004572872
29    impact             6.922918004572872
30    holding            6.922918004572872
31    hockey             6.922918004572872
32    plug               6.922918004572872
33    heels              6.922918004572872
34    handles            6.922918004572872
35    hankies            6.922918004572872
36    happiness          6.922918004572872
37    happy              6.922918004572872
38    harris             6.922918004572872
39    hatred             6.922918004572872
40    havilland          6.922918004572872
41    hayao              6.922918004572872
42    hayworth           6.922918004572872
43    heads              6.922918004572872
44    hearts             6.922918004572872
45    heartwarming               6.922918004572872
46    heche              6.922918004572872
47    heist              6.922918004572872
48    hilt               6.922918004572872
49    helen              6.922918004572872
```

In [137…

```python
row1=tfidf_transform(corpus2,vocab2,idf_array).toarray()[0] # document 0 of sparse m
print("Shape is: ",row1.shape,"\n\n")
print("First Document is: \n",row1)
```

```
Shape is:  (50,)


First Document is: 
 [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. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
```