TFIDF_Assignment

April 10, 2020

1 TFIDF Vectorizer

2 TASK 1

[1]:
```
import warnings
warnings.filterwarnings("ignore")
```

[2]:
```
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
```

[3]:
```
corpus = [
     'this is the first document',
     'this document is the second document',
     'and this is the third one',
     'is this the first document',
]
```

[4]:
```
#function to calculate idf value

def idf(word,dataset):
    N=len(dataset) #length of corpus
    No_of_docs=0
    for i in range(N):
        if word in dataset[i].split():  #splitting document by document in␣
 ↪corpus
            No_of_docs+=1     #word present in no of documents count
    if No_of_docs>0:
        idfvalue=1+math.log((1+N)/(1+No_of_docs))  #idf formula
        return idfvalue
    else:
        return 0
```

[5]:
```
def fit(dataset):
    unique_words = set()
    if isinstance(dataset, (list,)):
        for row in dataset:  # for each document in the dataset
            for word in row.split(" "):  # for each word in the document
                if len(word) < 2:
                    continue
                unique_words.add(word)
        unique_words = sorted(list(unique_words)) #unique words in corpus
        arr=[]
        for word in unique_words:
            arr.append(idf(word,corpus))   #calculating idf values for each␣
↪unique word
        list_idf=np.array(arr)
        vocab = {j:i for i,j in enumerate(unique_words)}
        return vocab,list_idf
    else:
        print("you need to pass list of sentance")
```

[6]:
```
def transform(dataset,vocab):
    rows = []
    columns = []
    values = []
    if isinstance(dataset, (list,)):
        for idx, row in enumerate(tqdm(dataset)):    #for each document in the␣
↪dataset
            word_freq = dict(Counter(row.split()))  #frequency of word in row
            for word, freq in word_freq.items():  # for each unique word in␣
↪corpus
                if len(word) < 2:
                    continue
                col_index = vocab.get(word,-1)  #retreving the dimension number␣
↪of a word, if not exists return -1
                if col_index !=-1:  #if word exists in vocab
                    rows.append(idx)              #storing row no and
                    columns.append(col_index)   #dimension no for getting␣
↪sparse matrix
                    idfvalue=list_idf[col_index]  #getting idf value from␣
↪list_idf
                    tf=float(freq/len(row.split())) #term frequency of word
                    values.append(tf*idfvalue) #calculating tfidf value
        return csr_matrix((values, (rows,columns)),␣
↪shape=(len(dataset),len(vocab)))

    else:
        print("you need to pass list of strings")
```

[7]:
```
vocab,list_idf= fit(corpus)
print(list(vocab.keys()))
print(list_idf)
```

```
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
[1.91629073 1.22314355 1.51082562 1.         1.91629073 1.91629073
 1.         1.91629073 1.        ]
```

[8]:
```
output=transform(corpus,vocab)
tfidf=normalize(output)
print(tfidf)
```

```
100%|
    | 4/4 [00:00<00:00, 799.87it/s]
```

```
  (0, 1)        0.4697913855799205
  (0, 2)        0.580285823684436
  (0, 3)        0.3840852409148149
  (0, 6)        0.3840852409148149
  (0, 8)        0.3840852409148149
  (1, 1)        0.6876235979836937
  (1, 3)        0.2810886740337529
  (1, 5)        0.5386476208856762
  (1, 6)        0.2810886740337529
  (1, 8)        0.2810886740337529
  (2, 0)        0.511848512707169
  (2, 3)        0.267103787642168
  (2, 4)        0.511848512707169
  (2, 6)        0.267103787642168
  (2, 7)        0.511848512707169
  (2, 8)        0.267103787642168
  (3, 1)        0.4697913855799205
  (3, 2)        0.580285823684436
  (3, 3)        0.3840852409148149
  (3, 6)        0.3840852409148149
  (3, 8)        0.3840852409148149
```

[9]:
```
print(tfidf[0])
```

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

[10]:
```
tfidf.shape
```

[10]: (4, 9)

[11]:
```
print(tfidf[0].toarray())
```

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

3 TASK 2

4 Implement max features functionality:

[12]:
```
import warnings
warnings.filterwarnings("ignore")
```

[13]:
```
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
```

[14]:
```
# Here corpus is of list type

import pickle
with open('cleaned_strings', 'rb') as f:
    corpus = pickle.load(f)

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

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

[15]:
```
#function to calculate idf value

def idf(word,dataset):
    N=len(dataset)   #length of corpus
    No_of_docs=0
    for i in range(N):
        if word in dataset[i].split():  #splitting document by document in␣
 ↪corpus
            No_of_docs+=1
    if No_of_docs>0:
        idfvalue=1+math.log((1+N)/(1+No_of_docs))
        return idfvalue
    else:
```

```
        return 0
```

[16]:
```
def fit(dataset):
    unique_words = set()
    if isinstance(dataset, (list,)):
        for row in dataset:   # for each document 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=list(unique_words) #unique words in corpus
        list2=[]
        for word in unique_words:
            list2.append(idf(word,corpus))  #calculating idf values for each␣
↪unique word
        list_1=np.array(unique_words)
        list_2=np.array(list2)
        index=np.argsort(list_2*-1)[:50]  #sorting idf_values array by index␣
↪and index_list  contains indices of top 50 idfvalues
        list_words=np.array(list_1)[index]  #getting top 50 idf words using␣
↪index and storing in list_words array
        list_idf=np.array(list_2)[index]  #getting top 50 idf values using␣
↪index and storing in list_idf array
        index2=np.argsort(list_words)            #sorting list_words␣
↪(alphabetic order) by index
        final_list=np.array(list_words)[index2]  #and storing in final_list␣
↪array
        final_idf=np.array(list_idf)[index2]     #and idfs of sorted list_words
        vocab={j:i for i,j in enumerate(final_list)}
        return vocab,final_idf
    else:
        print("you need to pass list of sentance")
```

[17]:
```
def transform(dataset,vocab):
    rows = []
    columns = []
    values = []
    if isinstance(dataset, (list,)):
        for idx, row in enumerate(tqdm(dataset)):  #for each document in the␣
↪dataset
            word_freq = dict(Counter(row.split()))  #frequency of word in row
            for word, freq in word_freq.items():   #for each unique word in␣
↪corpus
                if len(word) < 2:
                    continue
```

```
                col_index = vocab.get(word,-1)  #retreving the dimension number␣
↪of a word, if not exists return -1
                if col_index !=-1:  #if word exists in vocab
                    rows.append(idx)            #storing row no and
                    columns.append(col_index)   #dimension no for getting␣
↪sparse matrix
                    idfvalue=list_idf[col_index]  #getting idf value from␣
↪list_idf
                    tf=float(freq/len(row.split()))   #term frequency of word
                    values.append(idfvalue*tf)   #calculating tfidf value
        return csr_matrix((values, (rows,columns)),␣
↪shape=(len(dataset),len(vocab)))
    else:
        print("you need to pass list of strings")
```

[18]:
```
vocab,list_idf=fit(corpus)
print(list(vocab.keys()))
print(list_idf)
```

```
['admitted', 'aspects', 'atrocity', 'ben', 'bible', 'brevity', 'brings',
'brother', 'characterisation', 'commentary', 'cutie', 'dealt', 'delete',
'experiences', 'frances', 'hes', 'humans', 'indication', 'interacting', 'jack',
'kill', 'kristoffersen', 'laselva', 'layers', 'letting', 'logic', 'london',
'maker', 'murdering', 'oriented', 'patriotism', 'person', 'planned',
'politically', 'ponyo', 'preservation', 'relaxing', 'renowned', 'revenge',
'ryan', 'shakespears', 'struggle', 'student', 'suspension', 'teacher',
'unneeded', 'unrealistic', 'vehicles', 'versus', 'water']
[6.922918 6.922918 6.922918 6.922918 6.922918 6.922918 6.922918 6.922918
 6.922918 6.922918 6.922918 6.922918 6.922918 6.922918 6.922918 6.922918
 6.922918 6.922918 6.922918 6.922918 6.922918 6.922918 6.922918 6.922918
 6.922918 6.922918 6.922918 6.922918 6.922918 6.922918 6.922918 6.922918
 6.922918 6.922918 6.922918 6.922918 6.922918 6.922918 6.922918 6.922918
 6.922918 6.922918 6.922918 6.922918 6.922918 6.922918 6.922918 6.922918
 6.922918 6.922918]
```

[19]:
```
output=transform(corpus,vocab)
tfidf=normalize(output)
```

```
100%|
 | 746/746 [00:00<00:00, 39367.78it/s]
```

[20]:
```
tfidf.shape
```

[20]: (746, 50)

[21]:
```
print(tfidf[135])
```

```
  (0, 11)       0.3535533905932738
  (0, 20)       0.3535533905932738
  (0, 25)       0.3535533905932738
  (0, 30)       0.3535533905932738
  (0, 49)       0.7071067811865476
```

[22]:
```
print(tfidf[135].toarray())
```

```
[[0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.35355339
  0.         0.         0.         0.         0.         0.
  0.         0.         0.35355339 0.         0.         0.
  0.         0.35355339 0.         0.         0.         0.
  0.35355339 0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.70710678]]
```

[ ]: