Custom TFIDF

January 1, 2022

0.1 Task - 1

[2]:
```
from collections import Counter
from scipy.sparse import csr_matrix
import math
import operator
from sklearn.preprocessing import normalize
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer

def IDF(corpus, unique_words):
    """
    This function calculates and return the IDF values.
    """
    idf_dict={}
    N=len(corpus)
    for i in unique_words:
        count=0
        for sen in corpus:
            if i in sen.split():  # checks if the word from unique_word present␣
 ↪in the sentence
                count=count+1     # and increment if it is there
            idf_dict[i]=1+(math.log((1+N)/(count+1))) #IDF formula
    return idf_dict

def fit(dataset):
    """
    This function creates dict of unique words in the corpus(vocab) and returns␣
 ↪vocab and Idf values
    """
    unique_words = set()     # at first we will initialize an empty set
    if isinstance(dataset, (list,)):  # proceed if it is list
        for row in dataset:
            for word in row.split(" "):
                if len(word) < 2:   # removes if the word is only one letter
                    continue
                unique_words.add(word) # Add the word to the set
```

```
        unique_words = sorted(list(unique_words)) # convert set to list and␣
↪sort it
        vocab = {j:i for i,j in enumerate(unique_words)}   # dictionary of␣
↪unique words of the corpus

    Idfs = IDF(dataset,unique_words)
    return vocab, Idfs


def transform(dataset,vocabulary,idf_values):
    """
    This function calculates TF-IDF values and returns normalized sparse matrix␣
↪of those values.
    """
    sparse_matrix = csr_matrix( (len(dataset), len(vocabulary)), dtype=np.
↪float64)
    for row  in range(0,len(dataset)):
        sen_words=Counter(dataset[row].split())  # words are stored as keys and␣
↪their counts are stored as values.
        for word in dataset[row].split():
            if word in  list(vocabulary.keys()):
                tf_idf_value=(sen_words[word]/len(dataset[row].
↪split()))*(idf_values[word]) #Finding TF-IDF value
                sparse_matrix[row,vocabulary[word]]=tf_idf_value  # converting␣
↪to sparse matrix

    output = normalize(sparse_matrix, norm='l2', axis=1, copy=False,␣
↪return_norm=False)  #Normalizing the final output
    return output



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

#using SKLearn

vectorizer = TfidfVectorizer()
vectorizer.fit(corpus)
skl_output = vectorizer.transform(corpus)

#Custom Code
```

```
Vocabulary, idf_of_vocabulary = fit(corpus)

print("Features from SKL :-",vectorizer.get_feature_names())
print("Features from custom code:-",list(Vocabulary.keys()))

print("\n\nIDF values from SKL :-",vectorizer.idf_)
print("IDF values from custom code :-",list(idf_of_vocabulary.values()))

final_output=transform(corpus,Vocabulary,idf_of_vocabulary)

print("\n\nshape of sklearn tfidf vectorizer output: ",skl_output.shape)
print("shape of custom code final output: ",final_output.shape)

print("\n\nfinal output using SKL(First Row) :-",skl_output[0].sorted_indices())
print("\nfinal output using custom code(First Row) :-",final_output[0])

print("\n\nfinal output using SKL(First Row,Dense Matrix) :-",skl_output[0].
 ↪toarray())
print("\nfinal output using custom code(First Row,Dense Matrix) :
 ↪-",final_output[0].toarray())
```

```
Features from SKL :- ['and', 'document', 'first', 'is', 'one', 'second', 'the',
'third', 'this']
Features from custom code:- ['and', 'document', 'first', 'is', 'one', 'second',
'the', 'third', 'this']


IDF values from SKL :- [1.91629073 1.22314355 1.51082562 1.         1.91629073
1.91629073
 1.         1.91629073 1.        ]
IDF values from custom code :- [1.916290731874155, 1.2231435513142097,
1.5108256237659907, 1.0, 1.916290731874155, 1.916290731874155, 1.0,
1.916290731874155, 1.0]


shape of sklearn tfidf vectorizer output:  (4, 9)
shape of custom code final output:  (4, 9)


final output using SKL(First Row) :-   (0, 1)   0.46979138557992045
  (0, 2)        0.5802858236844359
  (0, 3)        0.38408524091481483
  (0, 6)        0.38408524091481483
  (0, 8)        0.38408524091481483

final output using custom code(First Row) :-   (0, 1)   0.4697913855799205
  (0, 2)        0.580285823684436
```

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


final output using SKL(First Row,Dense Matrix) :- [[0.         0.46979139
0.58028582 0.38408524 0.         0.
  0.38408524 0.         0.38408524]]

final output using custom code(First Row,Dense Matrix) :- [[0.
0.46979139 0.58028582 0.38408524 0.         0.
  0.38408524 0.         0.38408524]]
```

0.2 Task - 2

[4]:
```
def fit_50(dataset):
    """
    This function creates dict of unique words in the corpus(vocab) and returns␣
 ↪vocab and Idf values
    """
    top50_vocab = []
    unique_words = set()     # at first we will initialize an empty set
    if isinstance(dataset, (list,)):  # proceed if it is list
        for row in dataset:
            for word in row.split(" "):
                if len(word) < 2:   # removes if the word is only one letter
                    continue
                unique_words.add(word) # Add the word to the set
        unique_words = sorted(list(unique_words)) # convert set to list and␣
 ↪sort it

    Idfs = IDF(dataset,unique_words)

    #extracting the top 50 words based on IDF values
    sorted_idfs = {k: v for k, v in sorted(Idfs.items(), key=lambda item:␣
 ↪item[1],reverse=True)} # sorting in descending order
    top50_Idfs = {k: sorted_idfs[k] for k in list(sorted_idfs)[:50]}  #␣
 ↪selecting first 50 elements from sorted idfs
    for i in unique_words:           # getting the words with highest idf values
        if i in top50_Idfs.keys():
            top50_vocab.append(i)
    top50_vocab = {j:i for i,j in enumerate(top50_vocab)}   # list to dictionary
    return top50_vocab, top50_Idfs

import pickle
with open('cleaned_strings', 'rb') as f:
    corpus1 = pickle.load(f)
```

```
Vocabulary_50, idf_of_vocabulary_50 = fit_50(corpus1)

print("Top 50 Features:-",list(Vocabulary_50.keys()))
print("\nTop 50 IDF values:-",list(idf_of_vocabulary_50.values()))

final_output_50 = transform(corpus1,Vocabulary_50,idf_of_vocabulary_50)

print("\nshape of final output: ",final_output_50.shape)
print("\nfinal output(First Row) :-",final_output_50[0])
print("\nfinal output(First Row,Dense Matrix) :-",final_output_50[0].toarray())
```

```
Top 50 Features:- ['aailiyah', 'abandoned', 'abroad', 'abstruse', 'academy',
'accents', 'accessible', 'acclaimed', 'accolades', 'accurate', 'accurately',
'achille', 'ackerman', 'actions', 'adams', 'add', 'added', 'admins',
'admiration', 'admitted', 'adrift', 'adventure', 'aesthetically', 'affected',
'affleck', 'afternoon', 'aged', 'ages', 'agree', 'agreed', 'aimless', 'aired',
'akasha', 'akin', 'alert', 'alike', 'allison', 'allow', 'allowing', 'alongside',
'amateurish', 'amaze', 'amazed', 'amazingly', 'amusing', 'amust', 'anatomist',
'angel', 'angela', 'angelina']

Top 50 IDF values:- [6.922918004572872, 6.922918004572872, 6.922918004572872,
6.922918004572872, 6.922918004572872, 6.922918004572872, 6.922918004572872,
6.922918004572872, 6.922918004572872, 6.922918004572872, 6.922918004572872,
6.922918004572872, 6.922918004572872, 6.922918004572872, 6.922918004572872,
6.922918004572872, 6.922918004572872, 6.922918004572872, 6.922918004572872,
6.922918004572872, 6.922918004572872, 6.922918004572872, 6.922918004572872,
6.922918004572872, 6.922918004572872, 6.922918004572872, 6.922918004572872,
6.922918004572872, 6.922918004572872, 6.922918004572872, 6.922918004572872,
6.922918004572872, 6.922918004572872, 6.922918004572872, 6.922918004572872,
6.922918004572872, 6.922918004572872, 6.922918004572872, 6.922918004572872,
6.922918004572872, 6.922918004572872, 6.922918004572872, 6.922918004572872,
6.922918004572872, 6.922918004572872, 6.922918004572872, 6.922918004572872,
6.922918004572872, 6.922918004572872, 6.922918004572872]

shape of final output:  (746, 50)

final output(First Row) :-   (0, 30)    1.0

final output(First Row,Dense Matrix) :- [[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. 0. 0. 0. 0.
  0. 0.]]
```