A_3

October 16, 2020

1 SK-Learn Implementation

[23]:

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

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
vectorizer.fit(corpus)
skl_output = vectorizer.transform(corpus)

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

print(vectorizer.get_feature_names())


# 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_)

# shape of sklearn tfidf vectorizer output after applying transform method.

skl_output.shape

# sklearn tfidf values for first line of the above corpus.
# Here the output is a sparse matrix

print(skl_output[0])

# 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 printing it.
# Notice that this output is normalized using L2 normalization. sklearn does␣
 ↪this by default.

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

```
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
[1.91629073 1.22314355 1.51082562 1.         1.91629073 1.91629073
 1.         1.91629073 1.        ]
  (0, 8)        0.38408524091481483
  (0, 6)        0.38408524091481483
  (0, 3)        0.38408524091481483
  (0, 2)        0.5802858236844359
  (0, 1)        0.46979138557992045
[[0.         0.46979139 0.58028582 0.38408524 0.         0.
  0.38408524 0.         0.38408524]]
```

[ ]:

2 Your custom implementation:

3 Task-1

[32]:

```
# 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

## 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',
]
```

[57]:

```
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
from tqdm import tqdm
import os
# fit method is used to identify the unique words in the corpus and add␣
 ↪dimension to it in the dictionary-format.
def fit(data):
  if type(data) == type(list()):
      s = set()
      for i in range(len(data)): # Iterating over every row in the corpus and␣
 ↪finding the unique words of (length > 2)
          x = data[i].split()
          for j in range(len(x)):
              if len(x[j]) < 2:
                  continue
              s.add(x[j])
      d = {j:i for i,j in enumerate(sorted(s))} # d : ( keys = unique-words )␣
 ↪and (values = dimension-number)
      return d
  else:
        print("you need to pass list of sentance")
```

[34]:

```
def transform(corpus,vocab):
  idf_dict = dict() # keys = unique-words , values = number of documents␣
 ↪contain the corresponding unique-word.

  idf_ = list() # used for printing the idf values

  for word in vocab.keys(): # this for-loop is used to find number of documents␣
 ↪contain the corresponding unique-word.
    c=0
    for row in (corpus):
      if word in row:
        c+=1
    idf_dict[word] = c
    idf_.append(1+ math.log((1+len(corpus))/(c+1)))
  print('\n',idf_)
  rows = []
  cols = []
  vals = []
  print('\n','*'*50)
  for indx,row in enumerate(corpus):
    a = dict(Counter(row.split()))
    for word,freq in a.items():
      col_indx = vocab.get(word,-1)
      if col_indx != -1:
        rows.append(indx)
        cols.append(col_indx)
        tf = freq / sum(a.values())
        idf = 1 + (math.log((1+len(corpus))/(1 + idf_dict[word])))
        res = (tf * idf)
        #print(tf,'*',idf,'=',res)
        vals.append(res)
  b = csr_matrix((vals, (rows,cols)), shape=(len(corpus),len(vocab)))
  b = normalize(b,norm='l2')
  return b
```

[35]:

```
vocab = fit(corpus)
print(list(vocab.keys()))
print(transform(corpus,vocab).toarray())
```

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

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

 **************************************************
[[0.         0.46979139 0.58028582 0.38408524 0.         0.
  0.38408524 0.         0.38408524]
 [0.         0.6876236  0.         0.28108867 0.         0.53864762
  0.28108867 0.         0.28108867]
 [0.51184851 0.         0.         0.26710379 0.51184851 0.
  0.26710379 0.51184851 0.26710379]
 [0.         0.46979139 0.58028582 0.38408524 0.         0.
  0.38408524 0.         0.38408524]]
```

[72]:

[72]:

[ ]:

[ ]:

4 Task-2

[36]:

```
import pickle
import math
corpus = pickle.load(open("cleaned_strings", "rb"))
```

[68]:

```
def fit(data):
  if type(data) == type(list()):
      s = set()
      '''
      Finding the unique words in the corpus
      '''
      for i in range(len(data)):
          x = data[i].split()
          for j in range(len(x)):
              if len(x[j]) < 2:
                  continue
              s.add(x[j])

      d1 = dict() # d1: is used to store the unique-words as keys and ( values␣
↪= number of documents contains this word )
      fit.d2 = dict() # d2: ( keys = unique-words) and (values =␣
↪Inverse-document-freq values) in decending order of values

      for word in s: # this for-loop is used to find number of documents␣
↪contain the corresponding unique-word.
        c=0
        for row in (data):
          if word in row:
            c+=1
        d1[word] = c
        fit.d2[word] = 1 + (math.log((1+len(data)) / (1 + d1[word])))
      s1 = (sorted(fit.d2.items(), key=lambda x:x[1],reverse=True)[:50]) # top␣
↪50 words based on idf scores.
     # s1 = sorted(s1) # since the idf-scores for top-50 words are same, so␣
↪sorted these top-50 words in alphabetical order.
      #print(s1)
      s2 = [i[0] for i in s1]
      d = {j:i for i,j in enumerate((s2))}
      return d
  else:
        print("you need to pass list of sentance")
```

[69]:

```
def transform(corpus,vocab):
  rows = []
  cols = []
  vals = []
  for indx,row in enumerate(corpus):
    a = dict(Counter(row.split()))
    for word,freq in a.items():
      col_indx = vocab.get(word,-1)
      if col_indx != -1:
        rows.append(indx)
        cols.append(col_indx)
        tf = freq / sum(a.values())
        idf = fit.d2[word] # using the same dictionary (fit.d2) from fit method␣
↪where ( keys = unique-words) and (values = Inverse-document-freq scores) in␣
↪decending order of values
        #print(idf)
        res = float(tf * idf)
        #print(tf,'*',idf,'=',res)
        vals.append(res)
  b = csr_matrix((vals, (rows,cols)), shape=(len(corpus),len(vocab)))
  b = normalize(b,norm='l2')
  return b
```

[70]:

```
vocab = fit(corpus)
print(list(vocab.keys()))
print(transform(corpus,vocab).toarray())
```

```
['holding', 'shameful', 'landscapes', 'removing', 'secondary', 'revenge',
'politically', 'repeating', 'massive', 'cliff', 'kathy', 'rendering', 'virus',
'hayworth', 'fire', 'cutie', 'fanciful', 'reporter', 'boss', 'represents',
'sounded', 'regardless', 'portrayed', 'angelina', 'spy', 'quaid', 'applause',
'shell', 'drawn', 'angela', 'voyage', 'evidently', 'timing', 'truth',
'unlockable', 'smith', 'menacing', 'edward', 'murdering', 'merit', 'selections',
'females', 'recover', 'pledge', 'flicks', 'finest', 'washed', 'manages',
'colours', 'discovery']
[[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.]]
```

[72]:

[52]:

[52]:

[52]: 2

[36]:

[36]:

[36]:

[36]:

[36]:

[36]:

[36]:

[66]:

[72]:

[72]: