parth.pandey13103447@gmail.com_5

January 15, 2020

1 TF IDF implementation

1.1 Test Corpus

[1]:
```
## 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'
]
```

1.2 SkLearn Implementation

[3]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
vectorizer.fit(corpus)
skl_output = vectorizer.transform(corpus)
```

[4]:
```
# sklearn feature names, they are sorted in alphabetic order by default.
print(vectorizer.get_feature_names())
```

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

[5]:
```
# 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_)
```

```
[1.91629073 1.22314355 1.51082562 1.         1.91629073 1.91629073
 1.         1.91629073 1.        ]
```

[6]:
```
vectorizer.vocabulary_
```

[6]:
```
{'this': 8,
 'is': 3,
 'the': 6,
 'first': 2,
 'document': 1,
 'second': 5,
 'and': 0,
 'third': 7,
 'one': 4}
```

[8]:
```
# shape of sklearn tfidf vectorizer output after applying transform method.
skl_output.shape
```

[8]:
```
(4, 9)
```

[9]:
```
# sklearn tfidf values for first line of the above corpus.
# Here the output is a sparse matrix
print(skl_output)
```

```
  (0, 8)        0.38408524091481483
  (0, 6)        0.38408524091481483
  (0, 3)        0.38408524091481483
  (0, 2)        0.5802858236844359
  (0, 1)        0.46979138557992045
  (1, 8)        0.281088674033753
  (1, 6)        0.281088674033753
  (1, 5)        0.5386476208856763
  (1, 3)        0.281088674033753
  (1, 1)        0.6876235979836938
  (2, 8)        0.267103787642168
  (2, 7)        0.511848512707169
  (2, 6)        0.267103787642168
  (2, 4)        0.511848512707169
  (2, 3)        0.267103787642168
  (2, 0)        0.511848512707169
  (3, 8)        0.38408524091481483
  (3, 6)        0.38408524091481483
  (3, 3)        0.38408524091481483
  (3, 2)        0.5802858236844359
  (3, 1)        0.46979138557992045
```

[10]:
```
# 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())
```

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

1.3 Custom TF-IDF Implementation

1.3.1 Allowed Imports

[11]:
```
# 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
```

1.3.2 TF-IDF Code

[15]:
```
class personal_tfidf:

    def __init__(self):
        self.corpus = []
        self.vocab_ = {}
        self.tf_ = {}
        self.idf_ = {}
        self.idf = []

# The Fit Function
    def fit(self,corpus):
        count = 0
        word_list = set()
# Creating vocab dictionary for each document with key as word and
# value as their frequency
        for line in corpus:
            for word in line.split():
                word_list.add(word)
```

```
        self.vocab_ = {word:count for count,word in␣
↪enumerate(sorted(list(word_list)))}

    def transform(self,corpus):
        count = 1
        for line in corpus:
            temp = dict(Counter(line.split()))
            local_sum = sum(temp.values())
# Counting the word occurence in a document
            for key , val in temp.items():
                if key in self.idf_.keys():
                    self.idf_[key] += 1
                else:
                    self.idf_[key] = 1
# Updating the values of Counter to tf from frequency to tf values
                temp[key] = round(val/local_sum,8)
            self.corpus.append((temp))

# Calculating the idf values for each word
        for key in self.idf_.keys():
            self.idf_[key] = round(1 + math.log((len(corpus)+1)/(self.
↪idf_[key]+1)),8)

# Keeping the idf for the output in different variable
        for key in self.vocab_.keys():
            self.idf.append(self.idf_[key])

# Calculating the tf-idf values and also row column for the CSR matrix
        row = []
        value = []
        col = []
        for index in range(len(self.corpus)):
            for key , val in self.corpus[index].items():
                value.append(self.idf_[key] * val)
                row.append(index)
                col.append(self.vocab_[key])

# Normalizing the output created in the CSR matrix
        return  normalize(csr_matrix((value,(row,col)),
            shape=(len(corpus),
            len(self.vocab_.keys()))),norm='l2')

# Function to print the vocab values
    def get_feature_names(self):
        return list(self.vocab_.keys())

p = personal_tfidf()
```

```
p.fit(corpus)
output = p.transform(corpus)
```

[232]:
```
print(p.get_feature_names())
```

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

[233]:
```
print(p.idf)
```

```
[1.91629073, 1.22314355, 1.51082562, 1.0, 1.91629073, 1.91629073, 1.0,
1.91629073, 1.0]
```

[234]:
```
output.shape
```

[234]:
```
(4, 9)
```

[235]:
```
print(output)
```

```
  (0, 1)        0.46979138558088085
  (0, 2)        0.5802858228626505
  (0, 3)        0.3840852413282814
  (0, 6)        0.3840852413282814
  (0, 8)        0.3840852413282814
  (1, 1)        0.6876235869144148
  (1, 3)        0.28108867824349915
  (1, 5)        0.5386476284259701
  (1, 6)        0.28108867824349915
  (1, 8)        0.28108867824349915
  (2, 0)        0.5118485126000253
  (2, 3)        0.2671037878474866
  (2, 4)        0.5118485126000253
  (2, 6)        0.2671037878474866
  (2, 7)        0.5118485126000253
  (2, 8)        0.2671037878474866
  (3, 1)        0.46979138558088085
  (3, 2)        0.5802858228626505
  (3, 3)        0.3840852413282814
  (3, 6)        0.3840852413282814
  (3, 8)        0.3840852413282814
```

1.4 Custom TF-IDF with max_features

1.4.1 Importing the word corpus provided for the task

[16]:
```
# Below is the code to load the cleaned_strings pickle file provided
# 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
```

1.4.2 Max_Feature TF-IDF Code

[44]:
```
# Write your code here.
# Try not to hardcode any values.
# Make sure its well documented and readble with appropriate comments.


class max_idf_tfidf():


    def __init__(self,max_features=50):
        self.corpus = []
        self.vocab_ = {}
        self.tf_ = {}
        self.idf_ = {}
        self.idf = []
        self.max_features = max_features

    def fit(self,corpus):
        count = 0
# Creatinng a list of dictionaries for each document containg word and it's␣
 ↪count
        word_list = set()
        for line in corpus:
            temp = dict(Counter(line.split()))
            for word in line.split():
                word_list.add(word)
# Counting the word occurence in a document
            for key , val in temp.items():
                if key in self.idf_.keys():
```

```
                    self.idf_[key] += 1
                else:
                    self.idf_[key] = 1
# Calculating the idf values
        for key in self.idf_.keys():
            self.idf_[key] = round(1 + math.log((len(corpus)+1)/(self.
 ↪idf_[key]+1)),8)

# Now keeping the top 50 idf values only

# Sorting on values descending
        self.idf_ = dict(sorted(self.idf_.items(),key = lambda x:␣
 ↪(x[1],x[0]),reverse=True))

# Creating the original vocab dictionaries
        self.vocab = {word:count for count, word in␣
 ↪enumerate(sorted(list(word_list))) }

# Creating the vocab with top 50 words
        for key ,val in self.vocab.items():
            if key in list(self.idf_.keys())[:self.max_features]:
                self.vocab_[key] = val

# Creating the final IDF Values dictionary for top 50 idf values
        for key in self.vocab_.keys():
            if key in list(self.idf_.keys())[:self.max_features]:
                self.idf.append(self.idf_[key])


    def transform(self,corpus):
        count = 1

# Creating the tf ddictionary as list of dictionaries where each dictionary
# is the tf of each document
        for line in corpus:
            temp = dict(Counter(line.split()))
            local_sum = sum(temp.values())

# Updating the values of Counter to tf for
            for key , val in temp.items():
                temp[key] = round(val/local_sum,8)

# Creating main corpus with the top 50 words only
            list_keys = list(temp.keys())
            for key in list_keys:
                if key not in self.vocab_.keys():
                    temp.pop(key,None)
```

```
            if len(temp.keys()) > 0:
                self.corpus.append((temp))

        row = []
        value = []
        col = []
        for index in range(len(self.corpus)):
            for key , val in self.corpus[index].items():
                value.append(self.idf_[key] * val)
                row.append(index)
                col.append(self.vocab_[key])
#       Normalizing the output
        return  normalize(csr_matrix((value,(row,col)),
            shape=(len(self.corpus),
            len(self.vocab.keys()))),norm='l2')


    def get_feature_names(self):
        return list(self.vocab_.keys())
```

[47]:
```
p = max_idf_tfidf(max_features=50)
p.fit(corpus)
print(p.get_feature_names())
output = p.transform(corpus)
print(p.idf)
print(output[0].toarray())
```

```
['weariness', 'weaving', 'website', 'wedding', 'weight', 'welsh', 'went',
'whenever', 'whine', 'whites', 'whoever', 'wide', 'widmark', 'wife', 'wih',
'wild', 'william', 'willie', 'wily', 'win', 'wise', 'within', 'witticisms',
'woa', 'wondered', 'wong', 'wont', 'woo', 'worked', 'worry', 'worthless',
'worthwhile', 'worthy', 'wouldnt', 'woven', 'wow', 'wrap', 'writers', 'wrote',
'x', 'yardley', 'yawn', 'yelps', 'younger', 'youthful', 'youtube', 'yun', 'z',
'zillion', 'zombiez']
[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]
[[0. 0. 0. … 0. 0. 0.]]
```