python_custom_tfidf

July 26, 2020

[1]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from collections import Counter
from scipy.sparse import csr_matrix
from sklearn.preprocessing import normalize
from tqdm import tqdm

import numpy as np
import math
import operator
import warnings

warnings.filterwarnings("ignore")
```

[2]:
```
import pickle

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

print("Number of documents in corpus = ", len(corpus))
```

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

[3]:
```
from collections import OrderedDict

def fit(dataset, max=None):
    def Convert(tup, di):
        di = dict(tup)
        return di
    dictionary = {}
    vocabulary = Counter()
    if isinstance(dataset, list):
        for rows in dataset:
            vocabulary.update([i.lower() for i in rows.split(" ") if len(i)>=2])
        vocabulary = dict(vocabulary)
        if max is None:
            vocabulary = dict(OrderedDict(sorted(vocabulary.items(), key=lambda␣
 ↪t: t[0])))
```

```
        else:
            vocabulary = dict(OrderedDict(sorted(vocabulary.items(), key=lambda␣
↪t: t[1])))
            vocabulary = [(i,j) for i, j in vocabulary.items()][0:max]
            vocabulary = dict(OrderedDict(sorted(Convert(vocabulary,␣
↪dictionary).items(), key=lambda t: t[0])))
        return vocabulary

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

0.0.1 FORMULA

$TF(t) = \frac{\text{Number of times term t appears in a document}}{\text{Total number of terms in the document}}$.

$IDF(t) = 1 + \log_e \frac{1 + \text{Total number of documents in collection}}{1+\text{Number of documents with term t in it}}$.

[4]:
```
import math

def transform(dataset, vocab):
    sorted_vocab = list(vocab.keys())
    no_doc_WithTerms = dict.fromkeys(sorted_vocab, 0)
    words_idf = dict.fromkeys(sorted_vocab, 0)
    def column_index(term):
        try:
            var = sorted_vocab.index(term)
        except:
            var = -1
        return var
    rows, columns, values = [], [], []
    if isinstance(dataset, list):
        for idx, row in enumerate(dataset):
            word_freq = dict(Counter(row.split(" ")))
            for word, _ in word_freq.items():
                if len(word) <=1:
                    continue
                try:
                    no_doc_WithTerms[str(word)] += 1
                except:
                    pass
        for idx, row in enumerate(dataset):
            word_freq = dict(Counter(row.split(" ")))
            for word, freq in word_freq.items():
                if column_index(word) != -1:
                    rows.append(idx)
                    columns.append(column_index(word))
                    tf = freq / sum(list(word_freq.values()))
```

```
                    no_of_doc = 1 + len(dataset)
                    no_doc_WithTerm = 1 + no_doc_WithTerms[word]
                    idf = 1 + math.log(no_of_doc / float(no_doc_WithTerm))
                    words_idf[word] = idf
                    values.append(tf*idf)
        words_idf = dict(OrderedDict(sorted(words_idf.items(), key=lambda t:␣
↪t[0])))
        return normalize(csr_matrix( ((values),(rows,columns)),␣
↪shape=(len(dataset),len(vocab)))), words_idf
```

1 Test 1

[5]:
```
corpus1 = [
     '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(corpus1)
skl_output = vectorizer.transform(corpus1)

print(vectorizer.get_feature_names(), "\n\n")

print(vectorizer.idf_, "\n\n")

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

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

[6]:
```
vocab = fit(corpus1)
print(list(vocab.keys()), "\n\n")
sparse, idf = transform(corpus1, vocab)
print(list(idf.values()), "\n\n", sparse.todense()[0])
```

```
['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]]
```

2 TASK 1

[7]:
```
vectorizer = TfidfVectorizer()
vectorizer.fit(corpus)
skl_output = vectorizer.transform(corpus)

print(vectorizer.get_feature_names()[0:5], "\n\n")

print(vectorizer.idf_[0:10], "\n\n")

print(skl_output.todense()[0], "\n\n")
print(skl_output.todense().shape, "\n\n")

vocab = fit(corpus)
print(list(vocab.keys())[0:5], "\n\n")
sparse, idf = transform(corpus, vocab)
print(list(idf.values())[0:10], "\n\n", sparse.todense()[0], "\n\n", sparse.
 ↪todense().shape)
```

```
['aailiyah', 'abandoned', 'ability', 'abroad', 'absolutely']


[6.922918   6.922918   6.22977082 6.922918   5.31348009 6.922918
 6.5174529  6.922918   6.922918   6.922918  ]


[[0. 0. 0. … 0. 0. 0.]]


(746, 2886)


['aailiyah', 'abandoned', 'ability', 'abroad', 'absolutely']


[6.922918004572872, 6.922918004572872, 6.229770824012927, 6.922918004572872,
5.3134800921387715, 6.922918004572872, 6.517452896464707, 6.922918004572872,
6.922918004572872, 6.922918004572872]

 [[0. 0. 0. … 0. 0. 0.]]
```

```
(746, 2886)
```

3 TASK 2

[8]:
```
vectorizer = TfidfVectorizer()

vectorizer.fit(corpus)

skl_output = vectorizer.transform(corpus)

print(vectorizer.idf_[:50], "\n\n")

vocab = fit(corpus, max=50)

print(vocab, "\n\n")

sparse, idf = transform(corpus, vocab)

print(list(idf.values())[0:50], "\n\n", sparse.todense().shape, "\n\n", sparse.
↪todense()[0])
```

```
[6.922918   6.922918   6.22977082 6.922918   5.31348009 6.922918
 6.5174529  6.922918   6.922918   6.922918   6.922918   6.922918
 6.922918   6.922918   6.5174529  6.5174529  6.922918   6.922918
 6.5174529  6.22977082 3.97847903 5.53662364 6.922918   5.31348009
 4.67162621 6.22977082 6.22977082 5.21816991 6.922918   6.5174529
 6.922918   6.922918   6.22977082 6.922918   6.922918   6.922918
 6.00662727 6.922918   6.922918   6.5174529  6.22977082 6.922918
 6.922918   6.922918   6.5174529  6.5174529  6.922918   6.00662727
 6.922918   6.922918  ]


{'aimless': 1, 'artiness': 1, 'attempting': 1, 'aye': 1, 'baby': 1, 'buffet': 1,
'changing': 1, 'conception': 1, 'constructed': 1, 'content': 1, 'distressed': 1,
'doomed': 1, 'dozen': 1, 'drifting': 1, 'effort': 1, 'emptiness': 1, 'existent':
1, 'fill': 1, 'florida': 1, 'gerardo': 1, 'highest': 1, 'insane': 1, 'messages':
1, 'minor': 1, 'muppets': 1, 'nearly': 1, 'number': 1, 'occurs': 1, 'overdue':
1, 'owls': 1, 'person': 1, 'post': 1, 'practically': 1, 'properly': 1, 'pulls':
1, 'punches': 1, 'puzzle': 1, 'require': 1, 'rocks': 1, 'science': 1,
'screenplay': 1, 'solving': 1, 'structure': 1, 'superlative': 1, 'teacher': 1,
'th': 1, 'tightly': 1, 'tone': 1, 'unlockable': 1, 'vitally': 1}


[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]

 (746, 50)

 [[0.57735027 0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.57735027 0.
  0.         0.57735027 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.        ]]
```

[ ]: