== AI for Trading == Project \_5\_starter

import re

#(..)TODO-1

def get\_documents(text):

"""

Extract the documents from the text

Parameters

----------

text : str

The text with the document strings inside

Returns

-------

extracted\_docs : list of str

The document strings found in `text`

"""

# TODO: Implement

# print( type(text) )

ans = []

print( text)

print()

# ans = re.search('<DOCUMENT>(\*)</DOCUMENT>', text)

# beg = text.find('<DOCUMENT>') + 10

# end = text.find('</DOCUMENT>', beg)

beg = re.compile(r'<DOCUMENT>')

end = re.compile(r'</DOCUMENT>')

doc\_start\_is = [x.end() for x in beg.finditer(text)]

doc\_end\_is = [x.start() for x in end.finditer(text)]

for doc\_start\_i, doc\_end\_i in zip(doc\_start\_is, doc\_end\_is):

ans.append(text[doc\_start\_i:doc\_end\_i])

# ans = text.find("<DOCUMENT>")

print(type(ans), len(ans), ans)

print("-----------------------------------------")

return ans #AssertionError: Wrong len for output extracted\_docs. Got 28, expected 1

# OUTPUT extracted\_docs:

# ['This is before the test document<DOCUMENT>\nThis is inside the document\nThis is the text that should be copied</DOCUMENT>\nThis is after the document\nThis shouldn\t be included.']

# EXPECTED OUTPUT FOR extracted\_docs:

# ['\nThis is inside the document\nThis is the text that should be copied']

project\_tests.test\_get\_documents(get\_documents)

—

def get\_document\_type(doc):

"""

Return the document type lowercased

Parameters

----------

doc : str

The document string

Returns

-------

doc\_type : str

The document type lowercased

"""

# TODO: Implement

# for i in doc:

# print(i)

# print(doc.split("\n")[1][6:].lower()) #10-k

return doc.split("\n")[1][6:].lower()

project\_tests.test\_get\_document\_type(get\_document\_type)

================================================

TODO

from nltk.stem import WordNetLemmatizer

from nltk.corpus import wordnet

#TODO/TPCODE:Use Lemmatized library

# https://www.nltk.org/\_modules/nltk/stem/wordnet.html

def lemmatize\_words(words):

"""

Lemmatize words

Parameters

----------

words : list of str

List of words

Returns

-------

lemmatized\_words : list of str

List of lemmatized words

"""

# TODO: Implement

lemmatized\_words = list()

for word in words:

temp = WordNetLemmatizer().lemmatize(word,pos='v')

lemmatized\_words.append(temp)

# print(lemmatized\_words)

return lemmatized\_words

"""

INPUT words:

['cow', 'running', 'jeep', 'swimmers', 'tackle', 'throw', 'driven']

OUTPUT lemmatized\_words:

['cow', 'running', 'jeep', 'swimmers', 'tackle', 'throw', 'driven']

EXPECTED OUTPUT FOR lemmatized\_words:

['cow', 'run', 'jeep', 'swimmers', 'tackle', 'throw', 'drive']

"""

project\_tests.test\_lemmatize\_words(lemmatize\_words)

===============================================

from collections import defaultdict, Counter

from sklearn.feature\_extraction.text import CountVectorizer

def word\_count(str):

counts = dict()

words = str.split()

for word in words:

if word in counts:

counts[word] += 1

else:

counts[word] = 1

return counts

# print( word\_count('the quick brown fox jumps over the lazy dog.'))

-------------------------------------------------------------

TODO

def get\_bag\_of\_words(sentiment\_words, docs):

"""

Generate a bag of words from documents for a certain sentiment

Parameters

----------

sentiment\_words: Pandas Series

Words that signify a certain sentiment

docs : list of str

List of documents used to generate bag of words

Returns

-------

bag\_of\_words : 2-d Numpy Ndarray of int

Bag of words sentiment for each document

The first dimension is the document.

The second dimension is the word.

"""

"""

print("sentiment\_words",sentiment\_words,sentiment\_words[0])

# 0 one

# 1 last

# 2 second

# dtype: object one

# TODO: Implement

# from numpy import array

# ans = np.array([docs,sentiment\_words])

bag\_of\_words = np.zeros([len(docs), len(sentiment\_words)])

# print(bag\_of\_words) # 'numpy.ndarray'

# [[ 0. 0. 0.]

# [ 0. 0. 0.]

# [ 0. 0. 0.]]

# for sentence in docs:

# for sw in sentiment\_words:

# bag\_of\_words = print(sentence)

for i in range( len(docs) ):

wc\_dic = word\_count( docs[i] )

print("-------------")

print("===diction",i,"====")

print(wc\_dic)

print()

print("sentence", docs[i])

print()

for j in range(len(sentiment\_words)):

# print(i,j) # 0 0

# print(docs[i]) # this is a document

# print(sentiment\_words[j]) # one

# print("how many sentiment words in the sentence?")

print("sentiment\_words[j]",sentiment\_words[j]) #,

if sentiment\_words[j] in wc\_dic :

print("-in-")

print("Match",sentiment\_words[j],wc\_dic[sentiment\_words[j]])

bag\_of\_words[i][j] = wc\_dic[sentiment\_words[j]]

# bag\_of\_words[i][j] = 0.0#float(wc\_dic[sentiment\_words[j]])

#why failed here??? type!!!

# bag\_of\_words[i][j] = str(docs[i]) + str(sentiment\_words[j])

# ValueError: could not convert string to float: 'this is a documentone'

###########################################################

# TODO: Implement

"""

from collections import defaultdict, Counter

from sklearn.feature\_extraction.text import CountVectorizer

vec = CountVectorizer(vocabulary=sentiment\_words)

vectors = vec.fit\_transform(docs)

words\_list = vec.get\_feature\_names()

bag\_of\_words = np.zeros([len(docs), len(words\_list)])

for i in range(len(docs)):

bag\_of\_words[i] = vectors[i].toarray()[0]

# print(bag\_of\_words)

# print("--------")

# print(bag\_of\_words.astype(int))

#['this is a document', 'this is second document', 'last one']

# 0 one [[0 0 0]

# 1 last [0 0 1]

# 2 second [1 1 0]]

print("==============================")

print(type( bag\_of\_words.astype(int) ))

print(bag\_of\_words.astype(int) )

return bag\_of\_words.astype(int)

################################

=lean C/Pt=

from collections import defaultdict, Counter

from sklearn.feature\_extraction.text import CountVectorizer

vec = CountVectorizer(vocabulary=sentiment\_words)

vectors = vec.fit\_transform(docs)

return vectors.toarray()

project\_tests.test\_get\_bag\_of\_words(get\_bag\_of\_words)

—

==============================================

TODO

from sklearn.metrics import jaccard\_similarity\_score

# def sim\_jac(u,v):

# print("np.sum(np.minimum(u,v))", np.sum(np.minimum(u,v)) )

# print("np.sum(np.maximum(u,v)", np.sum(np.maximum(u,v)) )

# return np.sum(np.minimum(u,v))/np.sum(np.maximum(u,v))

def get\_jaccard\_similarity(bag\_of\_words\_matrix):

"""

Get jaccard similarities for neighboring documents

Parameters

----------

bag\_of\_words : 2-d Numpy Ndarray of int

Bag of words sentiment for each document

The first dimension is the document.

The second dimension is the word.

Returns

-------

jaccard\_similarities : list of float

Jaccard similarities for neighboring documents

"""

# TODO: Implement

"""

size = len( bag\_of\_words\_matrix )

matrix\_jac = np.zeros((size,size))

for i in range(size):

for j in range(size):

u = bag\_of\_words\_matrix[i]

v = bag\_of\_words\_matrix[j]

matrix\_jac[i][j] = sim\_jac(u,v)

# AssertionError: Wrong type for output jaccard\_similarities. Got <class 'numpy.ndarray'>, expected <class 'list'>

print("matrix\_jac",matrix\_jac)

# AssertionError: Wrong len for output jaccard\_similarities. Got 3, expected 2

return list(matrix\_jac)

"""

jaccard\_similarities = []

# print("bag\_of\_words\_matrix",bag\_of\_words\_matrix)

bool\_bow\_mat = bag\_of\_words\_matrix.astype(bool)

# print("bool\_bow\_mat",bool\_bow\_mat)

# compute jaccard similary for neighboring docs

for i in range(bag\_of\_words\_matrix.shape[0]-1):

jaccard\_similarities.append(jaccard\_similarity\_score(bool\_bow\_mat[i], bool\_bow\_mat[i+1]))

# print("jaccard\_similarities",jaccard\_similarities)

return jaccard\_similarities

project\_tests.test\_get\_jaccard\_similarity(get\_jaccard\_similarity)

==========================================================

TODO

from sklearn.feature\_extraction.text import TfidfVectorizer

def get\_tfidf(sentiment\_words, docs):

"""

Generate TFIDF values from documents for a certain sentiment

Parameters

----------

sentiment\_words: Pandas Series

Words that signify a certain sentiment

docs : list of str

List of documents used to generate bag of words

Returns

-------

tfidf : 2-d Numpy Ndarray of float

TFIDF sentiment for each document

The first dimension is the document.

The second dimension is the word.

"""

# TODO: Implement

print("sentiment\_words",sentiment\_words)

# 0 one

# 1 last

# 2 second

# dtype: object

print()

print("docs",docs)

# ['this is a document', 'this document is the second document', 'last one']

# vectorizer = TfidfVectorizer()

# AssertionError: Wrong shape for output tfidf. Got (3, 7), expected (3, 3)

vectorizer = TfidfVectorizer(vocabulary=sentiment\_words.values)

X = vectorizer.fit\_transform(docs)

print("X",X)

print("X.toarray()",X.toarray())

# AssertionError: Wrong type for output tfidf. Got <class 'scipy.sparse.csr.csr\_matrix'>, expected <class 'numpy.ndarray'>

return X.toarray()

project\_tests.test\_get\_tfidf(get\_tfidf)

##############################################################

import numpy as np

from math import \*

def \_tf(freq, avg, include\_log=True):

#(..)cp

if include\_log:

return 0 if freq == 0 else (1+log(freq))/(1+log(avg))

else:

return freq/avg

# compute idf

from collections import defaultdict

from math import log

def get\_idf(corpus, include\_log=True):

print("corpus:\n",corpus)

N = len(corpus)

print("N",N) #15

freq = defaultdict(int)

words = set()

for c in corpus:

print("len(c)",len(c), "len(set(c))", len(set(c)) ) #,c)

words |= set(c)

print("len of words",len(words)) # len of words 2554

for w in words:

freq[w] = sum([ w in c for c in corpus])

if include\_log:

return { w:log(N/freq[w]) for w in freq }

else:

return { w:N/freq[w] for w in freq }

##########################################################

def get\_tf(txt, include\_log=True):

# TO DO

# pass

#(..)cp

freq = bag\_of\_words(txt)

avg = np.mean(list(freq.values()))

tf = {w:\_tf(f,avg, include\_log) for w,f in freq.items()}

return defaultdict(int, tf)

##########################################################

def get\_vector(tf, idf):

# TO DO

# pass

#(..)copy

return np.array([ tf[w]\*idf[w] for w in idf ])

############################################################

############################################################

====================================================================

TODO

from sklearn.metrics.pairwise import cosine\_similarity

def get\_cosine\_similarity(tfidf\_matrix):

"""

Get cosine similarities for each neighboring TFIDF vector/document

Parameters

----------

tfidf : 2-d Numpy Ndarray of float

TFIDF sentiment for each document

The first dimension is the document.

The second dimension is the word.

Returns

-------

cosine\_similarities : list of float

Cosine similarities for neighboring documents

"""

# TODO: Implement

cosine\_similarities = []

print("tfidf\_matrix", tfidf\_matrix.shape,"\n",tfidf\_matrix)

print()

bool\_tfidf\_matrix = tfidf\_matrix.astype(bool)

print("bool\_tfidf\_matrix", bool\_tfidf\_matrix.shape,"\n",bool\_tfidf\_matrix)

print()

############################################################

cosine\_similarities = []

for i in range(len(tfidf\_matrix)-1):

cosine\_similarities.append(cosine\_similarity(tfidf\_matrix[i].reshape(1, -1),tfidf\_matrix[i+1].reshape(1, -1))[0,0])

return cosine\_similarities

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

# compute cosine similarities between neighboring docs

cosine\_similarities = list(np.diag(cosine\_similarity(tfidf\_matrix, tfidf\_matrix), k=1))

print("cosine\_similarities", cosine\_similarities)

return cosine\_similarities

project\_tests.test\_get\_cosine\_similarity(get\_cosine\_similarity)

==================================================================