STAT 29000 Project 4

Topics: python3, packages, pandas

Motivation: In order to become proficient with new skills, we need to practice. We will reinforce skills learned in the previous project to continue to solve problems. Often times you'll be tasked to try and understand someone's work and implement it in code. This project will do this.

Context: The previous project introduced an abundance of topics important to a solid python3 foundation. In this project we will continue to practice by solving data driven problems relating to our `media` package.

Scope: Writing functions, and using the `pandas` library to manipulate data.

First, we must install the missing package `stop-words`. In scholar, open up a shell and type the following:

```
python3.6 -m pip install stop-words --user
```

For the following questions, please copy and paste the following code at the top of your Jupyter notebook. We are simply importing useful Python modules that are required for this project. Be sure to replace "PURDUEALIAS" with your Purdue username. Note that if you get an error after running this chunk of code, you may need to click on Kernel->Restart.

```
# the following two lines tell notebook.scholar.rcac.purdue.edu's default python interpreter
# that it should look in ~/.local/lib/python3.6/site-packages for packages as well
# if you do not include these two lines at the very top of your notebook, you won't
# be able to import and use the packages we installed at earlier times/dates
import sys
sys.path.append("/home/PURDUEALIAS/.local/lib/python3.6/site-packages")

from random import sample
import string
from stop_words import get_stop_words
from math import log
```

You can find useful examples that walk you through relevant material here or on scholar: `/class/datamine/data/spring2020/s`
It is highly recommended to read through these to help solve problems.

Use the template found here or on scholar: `/class/datamine/data/spring2020/stat29000project04template.ipynb` to submit your solutions.

After each problem, we've provided you with a list of keywords. These keywords could be package names, functions, or important terms that will point you in the right direction when trying to solve the problem, and give you accurate terminology that you can further look into online. You are not required to utilize all the given keywords. You will receive full points as long as your code gives the correct result.

Question 1: tfidf

1a. *(2 pt)* Write a function called `tf` (short for term frequency) that accepts a *document* and a tuple of strings called *terms*. `tf` should return a list where each element in the list is the number of times that each *term* appears in the *document*. Note that we want this function to be case-insensitive, so "The" and "the" should be looked at as the same word. In addition, remove punctuation so 'farm.' below, becomes 'farm'. You can get a list of punctuation from the `string` package via `list(string.punctuation)`.

```
# a small example
my_terms = ('the', 'a', 'farm',)
my_document = 'I went to the farm and a boar charged at me. I will not return to the farm.'
```

```
tf(my_document, my_terms)
# [2, 1, 2]
```

Hint: *Be careful. Double check that your function is completely case-insensitive.*

Hint: *Within you function, I would suggest starting with a single for loop that loops on each term. This can also be accomplished using list comprehensions.*

Reminder: *Remember, tuples use (), lists use [], and sets use {}.*

Keywords: *python string methods: count, split, lower*

1b. *(2 pt)* Write a function called `idf` (short for inverse document frequency) that accepts a tuple of strings (aka documents) named *corpus* and a tuple of strings named *terms*. `idf` should return a list where each element in the list is the result of the following calculation for each term:

$$log_e\left(\frac{\#\ of\ documents\ in\ corpus}{\#\ of\ documents\ in\ corpus\ where\ term\ appears}\right)$$

Note that like the previous question, we want this function to be case-insensitive, so "The" and "the" should be looked at as the same word. In addition, remove punctuation so 'one.' below, becomes 'one'. You can get a list of punctuation from the `string` package via `list(string.punctuation)`.

Hint: *If you are struggling on how to begin – within the function, start by looping over each term in an outer loop. Within that outer loop, you will need to have another inner loop that loops on documents in the corpus (calculating the number of documents in the corpus that contain the current term (from our outer loop)).*

Hint: *A dummy example and output below.*

```
corpus = (
  "This is a sentence in the corpus.",
  "Each of these is a document in the corpus.",
  "Another sentence is here.",
  "The last sentence made no sense.",
  "Neither did that last one, another confusing one.",
  "Last one for sure.",
)

terms = ("sentence", "is", "a", "the", "that", "one", "last")

print(idf(corpus, terms))

[0.6931471805599453,
 0.6931471805599453,
 1.0986122886681098,
 0.6931471805599453,
 1.791759469228055,
 1.0986122886681098,
 0.6931471805599453]

# as of right now, this should create a ZeroDivisionError since
# "never" never occurs anywhere in any document in the corpus
terms = ("sentence", "is", "a", "the", "that", "one", "last", "never")
idf(corpus, terms)
```

Hint: *A potentially useful fact to know is if you `sum([True, True, False])` the result is 2.*

Keywords: *sum, math library, log function, python string methods: count, split, lower*

1c. *(1 pt)* Modify (1b) to accept a third argument *smooth* that is either a `True` or `False` value. By default, we want this argument to be set to `True` in order to prevent divide by zero errors. When `True`, it should return a list where each element in the list is the result of the following calculation for each term:

$$log_e\left(\frac{1+\#\ of\ documents\ in\ corpus}{1+\#\ of\ documents\ in\ corpus\ where\ term\ appears}\right)$$

otherwise, it should behave like (1b).

Hint: *A dummy example and output below.*

```
corpus = (
  "This is a sentence in the corpus.",
  "Each of these is a document in the corpus.",
  "Another sentence is here.",
  "The last sentence made no sense.",
  "Neither did that last one, another confusing one.",
  "Last one for sure.",
)

terms = ("sentence", "is", "a", "the", "that", "one", "last")

print(idf(corpus, terms))

[0.5596157879354227, 0.5596157879354227, 0.8472978603872037, 0.5596157879354227, 1.252762968495368, 0.8

# now this should work
terms = ("sentence", "is", "a", "the", "that", "one", "last", "never")
idf(corpus, terms)

[0.5596157879354227,
 0.5596157879354227,
 0.8472978603872037,
 0.5596157879354227,
 1.252762968495368,
 0.8472978603872037,
 0.5596157879354227,
 1.9459101490553132]
```

Keywords: *default function arguments*

Question 2: putting it all together

2a. *(2 pt)* Write a function called `tfidf` that accepts a tuple of strings (aka documents) named *corpus* and a tuple of strings named *terms*. `tfidf` should return a list of lists. The outer list should be the same length as the number of documents in *corpus*. Each inner list corresponds to a document d in the *corpus* and should be the same length as the number of terms in *terms*. Each value of the inner list corresponds to a term t and should be the result of the following calculation:

$$(\#\ of\ times\ term\ t\ appears\ in\ document\ d) * log_e\left(\frac{1+\#\ of\ documents\ in\ corpus}{1+\#\ of\ documents\ in\ corpus\ where\ term\ t\ appears}\right)$$

Hint: *The functions you wrote in (1a) and (1c) will be useful here.*

Hint: *If you are struggling on how to begin – start by calculating your inverse document frequency. This only needs to be done once. Then loop on every document in the corpus, and complete your calculations within that loop.*

Hint: *A dummy example and output below.*

```
corpus = [
  "This is a sentence in the corpus.",
  "Another sentence is here.",
  "The last sentence made no sense.",
  "Neither did that last one, another confusing one.",
  "Last one for sure.",
]

terms = ["sentence", "is", "a", "the", "that", "one", "last"]

tfidf(corpus, terms)

[[0.4054651081081644, 0.6931471805599453, 1.0986122886681098, 0.6931471805599453, 0.0, 0.0, 0.0],
 [0.4054651081081644, 0.6931471805599453, 0.0, 0.0, 0.0, 0.0, 0.0],
 [0.4054651081081644, 0.0, 0.0, 0.6931471805599453, 0.0, 0.0, 0.4054651081081644],
 [0.0, 0.0, 0.0, 0.0, 1.0986122886681098, 1.3862943611198906, 0.4054651081081644],
 [0.0, 0.0, 0.0, 0.0, 0.0, 0.6931471805599453, 0.4054651081081644]]
```

Keywords: *zip*

2b. *(1 pt)* Write a function called `parse_doc` that, given a document, strips all punctuation, breaks the document into individual, lowercase, words, resulting in a list. From that list, remove all stop words. To get a list of stop words, do the following:

```
from stop_words import get_stop_words
stop_words = get_stop_words('english')
```

Hint: *This is a good link to see some different strategies to remove punctuation.*

Hint: *A dummy example and output below.*

```
parse_doc("This is a test. Okay, we're set.")
# ['test', 'okay', 'set']
```

Keywords: *string.punctuation, translate, maketrans*

2c. *(1 pt)* Write a function called `corpus_terms` that, given a corpus, creates a list of terms where every word in every document is present a single time (that is, a collection of unique words). Use the function created in (2b) to clean the documents in the corpus prior to creating the unique list of terms.

Hint: *A dummy example and output below.*

```
corpus = [
  "This is a sentence in the corpus.",
  "Another sentence is here.",
  "The last sentence made no sense.",
  "Neither did that last one, another confusing one.",
  "Last one for sure.",
]

corpus_terms(corpus)

['another',
```

```
 'sentence',
 'neither',
 'corpus',
 'confusing',
 'one',
 'made',
 'sure',
 'sense',
 'last']
```

Keywords: *set, string join*

2d. *(1 pt)* Update both the `tf` and `idf` functions to use the `parse_doc` function created in (2b). Don't modify your original answers, just copy, paste, then modify the functions in new notebook cell(s).

Project Submission:

Submit your solutions for the project at this URL: https://classroom.github.com/a/v5I5Im00 using the instructions found in the GitHub Classroom instructions folder on Blackboard.

Important note: Make sure you submit your solutions in both .ipynb and .pdf formats. We've updated our instructions to include multiple ways to convert your .ipynb file to a .pdf on scholar. You can find a copy of the instructions on scholar as well: `/class/datamine/data/spring2020/jupyter.pdf`. If for some reason the script does not work, just submit the .ipynb.