Assignment4 (Score: 6.0 / 6.0)

1. Test cell (Score: 1.0 / 1.0)
2. Test cell (Score: 1.0 / 1.0)
3. Test cell (Score: 1.0 / 1.0)
4. Test cell (Score: 1.0 / 1.0)
5. Test cell (Score: 1.0 / 1.0)
6. Test cell (Score: 1.0 / 1.0)

Assignment 4 - Document Similarity & Topic Modelling¶

Part 1 - Document Similarity¶

For the first part of this assignment, you will complete the functions `doc_to_synsets` and `similarity_score` which will be used by `document_path_similarity` to find the path similarity between two documents.

The following functions are provided:

- **`convert_tag:`** converts the tag given by `nltk.pos_tag` to a tag used by `wordnet.synsets` . You will need to use this function in `doc_to_synsets` .
- **`document_path_similarity:`** computes the symmetrical path similarity between two documents by finding the synsets in each document us `doc_to_synsets` , then computing similarities using `similarity_score` .

You will need to finish writing the following functions:

- **`doc_to_synsets:`** returns a list of synsets in document. This function should first tokenize and part of speech tag the document using `nltk.word_tokenize` and `nltk.pos_tag` . Then it should find each tokens corresponding synset using `wn.synsets(token, wordnet_tag` The first synset match should be used. If there is no match, that token is skipped.
- **`similarity_score:`** returns the normalized similarity score of a list of synsets (s1) onto a second list of synsets (s2). For each synset in s1, fin synset in s2 with the largest similarity value. Sum all of the largest similarity values together and normalize this value by dividing it by the number largest similarity values found. Be careful with data types, which should be floats. Missing values should be ignored.

Once doc_to_synsets and similarity_score have been completed, submit to the autograder which will run a test to check that these functions are runni correctly.

Do not modify the functions `convert_tag` and `document_path_similarity` .

In [1]:

```
%%capture
import numpy as np
import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
nltk.download('omw-1.4')
from nltk.corpus import wordnet as wn
import pandas as pd
nltk.data.path.append("assets/")

def convert_tag(tag):
    """Convert the tag given by nltk.pos_tag to the tag used by wordnet.synsets"""

    tag_dict = {'N': 'n', 'J': 'a', 'R': 'r', 'V': 'v'}
    try:
        return tag_dict[tag[0]]
    except KeyError:
        return None
```

In [2]:

Student's answer

```
def doc_to_synsets(doc):
    """
    Returns a list of synsets in document.

    Tokenizes and tags the words in the document doc.
    Then finds the first synset for each word/tag combination.
    If a synset is not found for that combination it is skipped.

    Args:
        doc: string to be converted

    Returns:
        list of synsets

    Example:
        doc_to_synsets('Fish are nvqjp friends.')
        Out: [Synset('fish.n.01'), Synset('be.v.01'), Synset('friend.n.01')]
    """
    tokens = nltk.word_tokenize(doc)
    pos = nltk.pos_tag(tokens)
    tags = [tag[1] for tag in pos]
    wntag = [convert_tag(tag) for tag in tags]
    ans = list(zip(tokens,wntag))
    sets = [wn.synsets(x,y) for x,y in ans]
    final = [val[0] for val in sets if len(val) > 0]

    return final


def similarity_score(s1, s2):
    """
    Calculate the normalized similarity score of s1 onto s2

    For each synset in s1, finds the synset in s2 with the largest similarity value.
    Sum of all of the largest similarity values and normalize this value by dividing it by the
    number of largest similarity values found.

    Args:
        s1, s2: list of synsets from doc_to_synsets

    Returns:
        normalized similarity score of s1 onto s2

    Example:
        synsets1 = doc_to_synsets('I like cats')
        synsets2 = doc_to_synsets('I like dogs')
        similarity_score(synsets1, synsets2)
        Out: 0.73333333333333339
    """

    s=[]
    for i1 in s1:
        r=[]
        scores=[x for x in [i1.path_similarity(i2) for i2 in s2]if x is not None]
        if scores:
            s.append(max(scores))
    # Your Code Here

    return sum(s)/len(s)# Your Answer Here
```

In [3]:

```
def document_path_similarity(doc1, doc2):
    """Finds the symmetrical similarity between doc1 and doc2"""

    synsets1 = doc_to_synsets(doc1)
    synsets2 = doc_to_synsets(doc2)

    return (similarity_score(synsets1, synsets2) + similarity_score(synsets2, synsets1)) / 2
```

In [4]:

```
document_path_similarity('I like cat', 'I like dog')
```

Out[4]:

```
0.7333333333333334
```

In [5]:

Grade cell: `cell-8513adbb001100a6` Score: 1.0 / 1.0

`paraphrases` is a DataFrame which contains the following columns: `Quality`, `D1`, and `D2`.

`Quality` is an indicator variable which indicates if the two documents `D1` and `D2` are paraphrases of one another (1 for paraphrase, 0 for not paraphrase).

In [6]:

```
# Use this dataframe for questions most_similar_docs and label_accuracy
paraphrases = pd.read_csv('assets/paraphrases.csv')
paraphrases.head()
```

Out[6]:

	Quality	D1	D2
0	1	Ms Stewart, the chief executive, was not expec...	Ms Stewart, 61, its chief executive officer an...
1	1	After more than two years' detention under the...	After more than two years in detention by the ...
2	1	"It still remains to be seen whether the reven...	"It remains to be seen whether the revenue rec...
3	0	And it's going to be a wild ride," said Allan ...	Now the rest is just mechanical," said Allan H...
4	1	The cards are issued by Mexico's consulates to...	The card is issued by Mexico's consulates to i...

most_similar_docs¶

Using `document_path_similarity`, find the pair of documents in paraphrases which has the maximum similarity score.

This function should return a tuple `(D1, D2, similarity_score)`

In [7]:

Student's answer

```
def most_similar_docs():

    similarities = [(paraphrase['D1'], paraphrase['D2'], document_path_similarity(paraphrase['D1'], paraphrase['D2'])
                     for index, paraphrase in paraphrases.iterrows()]
    similarity = max(similarities, key=lambda item:item[2])

    return similarity
most_similar_docs()
```

Out[7]:

```
('"Indeed, Iran should be put on notice that efforts to try to remake Iraq in their image will be aggressively put dow
 '"Iran should be on notice that attempts to remake Iraq in Iran\'s image will be aggressively put down," he said.\n',
 0.9590643274853801)
```

In [8]:

Grade cell: `cell-3fedb973b11cff11` Score: 1.0 / 1.0

label_accuracy¶

Provide labels for the twenty pairs of documents by computing the similarity for each pair using `document_path_similarity`. Let the classifier rule that if the score is greater than 0.75, label is paraphrase (1), else label is paraphrase (0). Report accuracy of the classifier using scikit-learn's accuracy_

This function should return a float.

In [9]:

Student's answer

```
def label_accuracy():
    from sklearn.metrics import accuracy_score
    y_true = paraphrases['Quality']
    y_pred = pd.Series([document_path_similarity(D1, D2) for D1, D2 in zip(paraphrases['D1'], paraphrases['D2'])]).ap

    return accuracy_score(y_true,y_pred)

label_accuracy()
```

Out[9]:

```
0.7
```

In [10]:

Grade cell: `cell-38683a0be95791b4` Score: 1.0 / 1.0

Part 2 - Topic Modelling¶

For the second part of this assignment, you will use Gensim's LDA (Latent Dirichlet Allocation) model to model topics in `newsgroup_data`. You will fir need to finish the code in the cell below by using gensim.models.ldamodel.LdaModel constructor to estimate LDA model parameters on the corpus, a save to the variable `ldamodel`. Extract 10 topics using `corpus` and `id_map`, and with `passes=25` and `random_state=34`.

In [11]:

```
import pickle
import gensim
from sklearn.feature_extraction.text import CountVectorizer

# Load the list of documents
with open('assets/newsgroups', 'rb') as f:
    newsgroup_data = pickle.load(f)

# Use CountVectorizor to find three letter tokens, remove stop_words,
# remove tokens that don't appear in at least 20 documents,
# remove tokens that appear in more than 20% of the documents
vect = CountVectorizer(min_df=20, max_df=0.2, stop_words='english',
                       token_pattern='(?u)\\b\\w\\w\\w+\\b')
# Fit and transform
X = vect.fit_transform(newsgroup_data)

# Convert sparse matrix to gensim corpus.
corpus = gensim.matutils.Sparse2Corpus(X, documents_columns=False)

# Mapping from word IDs to words (To be used in LdaModel's id2word parameter)
id_map = dict((v, k) for k, v in vect.vocabulary_.items())
```

In [12]:

Student's answer

```
# Use the gensim.models.ldamodel.LdaModel constructor to estimate
# LDA model parameters on the corpus, and save to the variable `ldamodel`

# Your code here:
ldamodel = gensim.models.ldamodel.LdaModel(corpus, id2word=id_map, num_topics=10, passes=25, random_state=34)
```

lda_topics¶

Using `ldamodel`, find a list of the 10 topics and the most significant 10 words in each topic. This should be structured as a list of 10 tuples where eac tuple takes on the form:

`(9, '0.068*"space" + 0.036*"nasa" + 0.021*"science" + 0.020*"edu" + 0.019*"data" + 0.017*"shuttle" + 0.015*"launch" + 0.015*"available" + 0.014*"center" + 0.013*"information"')`

for example.

This function should return a list of tuples.

In [13]:

Student's answer

```
def lda_topics():

    return ldamodel.print_topics(num_topics=10)
lda_topics()
```

Out[13]:

```
[(0,
  '0.056*"edu" + 0.043*"com" + 0.033*"thanks" + 0.022*"mail" + 0.021*"know" + 0.020*"does" + 0.014*"info" + 0.012*"mon
 (1,
  '0.024*"ground" + 0.018*"current" + 0.018*"just" + 0.013*"want" + 0.013*"use" + 0.011*"using" + 0.011*"used" + 0.010
 (2,
  '0.061*"drive" + 0.042*"disk" + 0.033*"scsi" + 0.030*"drives" + 0.028*"hard" + 0.028*"controller" + 0.027*"card" + 0
 (3,
  '0.023*"time" + 0.015*"atheism" + 0.014*"list" + 0.013*"left" + 0.012*"alt" + 0.012*"faq" + 0.012*"probably" + 0.011
 (4,
  '0.025*"car" + 0.016*"just" + 0.014*"don" + 0.014*"bike" + 0.012*"good" + 0.011*"new" + 0.011*"think" + 0.010*"year"
 (5,
  '0.030*"game" + 0.027*"team" + 0.023*"year" + 0.017*"games" + 0.016*"play" + 0.012*"season" + 0.012*"players" + 0.01
 (6,
  '0.017*"information" + 0.014*"help" + 0.014*"medical" + 0.012*"new" + 0.012*"use" + 0.012*"000" + 0.012*"research" +
 (7,
  '0.022*"don" + 0.021*"people" + 0.018*"think" + 0.017*"just" + 0.012*"say" + 0.011*"know" + 0.011*"does" + 0.011*"go
 (8,
  '0.034*"use" + 0.023*"apple" + 0.020*"power" + 0.016*"time" + 0.015*"data" + 0.015*"software" + 0.012*"pin" + 0.012*
 (9,
  '0.068*"space" + 0.036*"nasa" + 0.021*"science" + 0.020*"edu" + 0.019*"data" + 0.017*"shuttle" + 0.015*"launch" + 0.
```

In [14]:

Grade cell: `cell-be3408e13d501ecb` Score: 1.0 / 1.0

topic_distribution¶

For the new document `new_doc`, find the topic distribution. Remember to use vect.transform on the the new doc, and Sparse2Corpus to convert the sparse matrix to gensim corpus.

This function should return a list of tuples, where each tuple is `(#topic, probability)`

In [15]:

```
new_doc = ["\n\nIt's my understanding that the freezing will start to occur because \
of the\ngrowing distance of Pluto and Charon from the Sun, due to it's\nelliptical orbit. \
It is not due to shadowing effects. \n\n\nPluto can shadow Charon, and vice-versa.\n\nGeorge \
Krumins\n-- "]
```

In [16]:

Student's answer

```
def topic_distribution():

    new_doc_transformed = vect.transform(new_doc)
    corpus = gensim.matutils.Sparse2Corpus(new_doc_transformed, documents_columns=False)
    doc_topics = ldamodel.get_document_topics(corpus)
    topic_dist = []
    for val in list(doc_topics):
        for v in val:
            topic_dist.append(v)
    return topic_dist

topic_distribution()
```

Out[16]:

```
[(0, 0.020003108),
 (1, 0.020003324),
 (2, 0.020001281),
 (3, 0.49674824),
 (4, 0.020004038),
 (5, 0.020004129),
 (6, 0.020002972),
 (7, 0.020002645),
 (8, 0.020003129),
 (9, 0.34322715)]
```

In [17]:

Grade cell: `cell-9a7e35944a145c49` Score: 1.0 / 1.0

topic_names¶

From the list of the following given topics, assign topic names to the topics you found. If none of these names best matches the topics you found, crea
new 1-3 word "title" for the topic.

Topics: Health, Science, Automobiles, Politics, Government, Travel, Computers & IT, Sports, Business, Society & Lifestyle, Religion, Education.

This function should return a list of 10 strings.

In [18]:

Student's answer

```
def topic_names():

    return ["Education", "Automobiles", "Computers & IT", "Religion", "Automobiles", "Sports", "Health", "Religion",
topic_names()
```

Out[18]:

```
['Education',
 'Automobiles',
 'Computers & IT',
 'Religion',
 'Automobiles',
 'Sports',
 'Health',
 'Religion',
 'Computers & IT',
 'Science']
```

In [19]:

Grade cell: `cell-78ea383cc25da8da` Score: 1.0 / 1.0

This assignment was graded by mooc_adswpy:5a1483384bca, v1.47.103123