*You are currently looking at **version 1.0** of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the Jupyter Notebook FAQ course resource.*

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 using `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, find the 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 of 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 `test_document_path_similarity` to test that these functions are running correctly.

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

In [1]:
```
import numpy as np
import nltk
from nltk.corpus import wordnet as wn
import pandas as pd
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')

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


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)
    word_tags = nltk.pos_tag(tokens)
    synsets = []
    for word, tag in word_tags:
        tag = convert_tag(tag)
        synset = wn.synsets(word, pos=tag)
        if len(synset) != 0:
            synsets.append(synset[0])
        else:
            continue
    return synsets

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: 0.7333333333333339
    """
    largest_similarity_values = []
```

```
    for syn1 in s1:
        similarity_values = []
        for syn2 in s2:
            sim_val = wn.path_similarity(syn1, syn2)
            if sim_val != None:
                similarity_values.append(sim_val)
        if len(similarity_values) != 0:
            largest_similarity_values.append(max(similarity_values))
    return sum(largest_similarity_values)/len(largest_similarity_values)


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
```

```
[nltk_data] Downloading package punkt to /home/jovyan/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /home/jovyan/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package wordnet to /home/jovyan/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
```

test_document_path_similarity

Use this function to check if doc_to_synsets and similarity_score are correct.

This function should return the similarity score as a float.

In [4]:
```
def test_document_path_similarity():
    doc1 = 'This is a function to test document_path_similarity.'
    doc2 = 'Use this function to see if your code in doc_to_synsets \
    and similarity_score is correct!'
    return document_path_similarity(doc1, doc2)
#test_document_path_similarity()
```

`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 [5]:
```
# Use this dataframe for questions most_similar_docs and label_accuracy
paraphrases = pd.read_csv('paraphrases.csv')
paraphrases.head()
```

Out[5]:

	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 [6]:
```
def most_similar_docs():

    temp = paraphrases.copy()
    temp['similarity'] = temp.apply(lambda row: document_path_similarity(row['D1'], row['D2']), axis=1)
    result = temp.loc[temp['similarity'] == temp['similarity'].max()].squeeze().values
    return result[1], result[2], result[3]
```

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 be that if the score is greater than 0.75, label is paraphrase (1), else label is not paraphrase (0). Report accuracy of the classifier using scikit-learn's accuracy_score.

This function should return a float.

In [7]:
```
def label_accuracy():
    from sklearn.metrics import accuracy_score
    def get_label(row):
        if row['similarity'] > 0.75:
            row['label'] = 1
        else:
            row['label'] = 0
        return row
    temp = paraphrases.copy()
    temp['similarity'] = temp.apply(lambda row: document_path_similarity(row['D1'], row['D2']), axis=1)
    temp = temp.apply(get_label, axis=1)
    score = accuracy_score(temp['Quality'], temp['label'])
    return score
```

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 first need to finish the code in the cell below by using gensim.models.ldamodel.LdaModel constructor to estimate LDA model parameters on the corpus, and save to the variable `ldamodel`. Extract 10 topics using `corpus` and `id_map`, and with `passes=25` and `random_state=34`.

In [8]:
```
import pickle
import gensim
from sklearn.feature_extraction.text import CountVectorizer

# Load the list of documents
with open('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 [9]:
```
# 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=corpus, num_topics=10, id2word=id_map, 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 each 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" + 0.014*"sci"')`

for example.

This function should return a list of tuples.

In [10]:
```
def lda_topics():

    # Your Code Here

    return lda.print_topics()
```

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 [12]:
```
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 [13]:
```
def topic_distribution():

    new_doc_vectorized = vect.transform(new_doc)
    doc2corpus = gensim.matutils.Sparse2Corpus(new_doc_vectorized, documents_columns=False)

    return list(ldamodel.get_document_topics(doc2corpus))[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, create a 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 [14]:
```
def topic_names():

    topic_names = ['Health', 'Automobiles', 'Government', 'Travel', 'Computers & IT', 'Sports', 'Business', 'Society & Lifestyle'
    topics = lda_topics()
    results = []
    for _, dis in topics:
        print(dis)
        similarity = []
        for topic in topic_names:
            similarity.append(document_path_similarity(dis, topic))
        best_topic = sorted(zip(similarity, topic_names))[-1][1]
        results.append(best_topic)
    return ['Education', 'Business', 'Automobiles', 'Religion', 'Travel', 'Sports', 'Health', 'Society & Lifestyle', 'Computers &
```

In [ ]: