# **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: 0.73333333333333339  """  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.*

| ***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).

| *# Use this dataframe for questions most\_similar\_docs and label\_accuracy* paraphrases = pd.read\_csv('paraphrases.csv') paraphrases.head() |
| --- |

|  | Qaulity | Document 1 | Document 2 |
| --- | --- | --- | --- |
| 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)*

| **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.*

| **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.

| **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 [7]: *# 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.*

| **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)*

| 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\n elliptical orbit. \ It is not due to shadowing effects. \n\n\nPluto can shadow Charon, and vice-versa.\n\nGeorge \Krumins\n-- "]  In [10]: **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.*

| **def** **topic\_names**():    topic\_names = ['Health', 'Automobiles', 'Government', 'Travel', 'Computers & IT', 'Sports', 'Business', 'Society & Lifestyle', 'Region', 'Education']  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' |
| --- |