```
import os
root = '/home/tuomas/Python/DATA.STAT.840/Exam/wizardofoz'

def load_data_local(directory):
    book_text = []
    files = os.listdir(directory)
    for file in files:
        with open(directory +'/'+ file, 'r') as f:
            book_text.append(f.read())

    return book_text, files

book_texts_temp, files = load_data_local(root)
book_texts = []
for btt in book_texts_temp:
    book_texts.append(btt.split())
#%%
''' Part a) starts '''
#%%
import gensim
gensim_docs = book_texts
gensim_dictionary=gensim.corpora.Dictionary(gensim_docs)
# Create the document-term vectors
gensim_docvectors=[]
for k in range(len(gensim_docs)):
    docvector=gensim_dictionary.doc2bow(gensim_docs[k])
    gensim_docvectors.append(docvector)

#%% Run the LDA optimization
numtopics=5
randomseed=124574527
numiters=10000
ninits=5
gensim_ldamodel=gensim.models.ldamodel.LdaModel(gensim_docvectors,
                                                 id2word=gensim_dictionary,
                                                 num_topics=numtopics,
                                                 iterations=numiters,
                                                 random_state=randomseed)

#%% Get topic content: term-topic probabilities
import numpy
gensim_termtopicprobabilities=gensim_ldamodel.get_topics()
# Get topic prevalences per document, and overall topic prevalences
# (expected amount of documents per topic)
overallstrengths=numpy.zeros((numtopics,1))
documentstrengths=numpy.zeros((len(gensim_docvectors),numtopics))
for k in range(len(gensim_docvectors)):

topicstrengths=gensim_ldamodel.get_document_topics(gensim_docvectors[k],minimum_probabilit
y=0)
    for m in range(len(topicstrengths)):
        documentstrengths[k][topicstrengths[m][0]]=topicstrengths[m][1]
        overallstrengths[topicstrengths[m][0]]=\
        overallstrengths[topicstrengths[m][0]]+topicstrengths[m][1]
```

```
for topic in range(numtopics):
    print('Topic {}:'.format(topic))
    print(gensim_ldamodel.show_topic(topic,topn=20))
    print()

#%%
''' Part b) starts '''
#%% Create unique vocabularities from the texts

import numpy
# Find the vocabulary, in a distributed fashion
vocabularies=[]
indices_in_vocabularies=[]
# Find the vocabulary of each document
for book_lemmatizedtext in book_texts:
    # Get unique words and where they occur
    temptext = book_lemmatizedtext
    uniqueresults = numpy.unique(temptext,return_inverse=True)
    uniquewords = uniqueresults[0]
    wordindices = uniqueresults[1]

    # Store the vocabulary and indices of document words in it
    vocabularies.append(uniquewords)
    indices_in_vocabularies.append(wordindices)

#%% Create a unified vocabulary from the ebooks

# Unify the vocabularies.
# First concatenate all vocabularies
tempvocabulary = []
for k in range(len(book_texts)):
    tempvocabulary.extend(vocabularies[k])

# Find the unique elements among all vocabularies
uniqueresults = numpy.unique(tempvocabulary,return_inverse=True)
unifiedvocabulary = uniqueresults[0]
wordindices = uniqueresults[1]

# Translate previous indices to the unified vocabulary.
# Must keep track where each vocabulary started in
# the concatenated one.
vocabularystart = 0
myindices_in_unifiedvocabulary = []
for k in range(len(book_texts)):
    # In order to shift word indices, we must temporarily
    # change their data type to a Numpy array
    tempindices = numpy.array(indices_in_vocabularies[k])
    tempindices = tempindices + vocabularystart
    tempindices = wordindices[tempindices]
    myindices_in_unifiedvocabulary.append(tempindices)
    vocabularystart += len(vocabularies[k])
```

```
#%% Create TF-IDF vectors
import scipy.sparse
remainingvocabulary = unifiedvocabulary
n_docs=len(book_texts)
n_vocab=len(remainingvocabulary)
# Matrix of term frequencies
tfmatrix=numpy.empty(shape=(n_docs,n_vocab)) #scipy.sparse.lil_matrix((n_docs,n_vocab))
# Row vector of document frequencies
dfvector=numpy.zeros(n_vocab)#scipy.sparse.lil_matrix((1,n_vocab))
# Loop over documents
for k in range(n_docs):
    print(k)
    # Row vector of which words occurred in this document
    temp_dfvector=scipy.sparse.lil_matrix((1,n_vocab))
    # Loop over words
    for l in range(len(book_texts[k])):
        # Add current word to term-frequency count and document-count
        print(l)
        currentword=myindices_in_unifiedvocabulary[k][l]
        tfmatrix[k,currentword] += 1
        temp_dfvector[0,currentword] = 1
    # Add which words occurred in this document to overall document counts
    dfvector += temp_dfvector

#%%
# Use the count statistics to compute the tf-idf matrix
tfidfmatrix=numpy.empty(shape=(n_docs,n_vocab))# scipy.sparse.lil_matrix((n_docs,n_vocab))
# Let's use raw term count, and smoothed logarithmic idf
dfvector = numpy.squeeze(numpy.asarray(dfvector))
idfvector=numpy.log(1+((dfvector.ravel()+1)**-1)*n_docs)

for k in range(n_docs):
    # Combine the tf and idf terms
    tfidfmatrix[k,:]=tfmatrix[k,:]*idfvector

#%%
import sys
files = numpy.array(files)
doc1 = numpy.where(files == 'paragraph000.txt')[0][0]
doc2 = numpy.where(files == 'paragraph100.txt')[0][0]
doc3 = numpy.where(files == 'paragraph200.txt')[0][0]
docs = [doc1,doc2,doc3]

def cos_sim(x1, x2):
    num = numpy.inner(x1, x2)
    denom = numpy.linalg.norm(x1)*numpy.linalg.norm(x2)
    return num/denom

closestdists = []
closestidxs = []
for docidx in docs:
    closest = sys.float_info.max
    closesidx = 0
    for i in range(tfidfmatrix.shape[0]):
        if i==docidx: continue
        dist = cos_sim(tfidfmatrix[docidx], tfidfmatrix[i])
        if dist < closest:
            closest = dist
            closesidx = i

    closestdists.append(dist)
    closestidxs.append(closesidx)

docs = ['paragraph000.txt','paragraph100.txt','paragraph200.txt']
for i in range(3):
    print('Closest to {} = idx {}'.format(docs[i], closestidxs[i]))

#%%
```