Exercise 7 - Vector space model P2

First name: Brian

Last name: Schweigler

Matriculation number: 16-102-071

Q1: For each genre, generate a "profile" in the form of a single vector representing the entire set of plays corresponding to this genre. Build such a profile for each of the three genres (Comedy, Tragedy and Tragicomedy).

Can take this from the solutions of last series:

In [1]:
```
%load_ext autoreload
%autoreload 2
%matplotlib inline

import matplotlib.pyplot as plt
import pandas as pd
import re
import numpy as np
import lxml.etree
import os
from scipy import stats
import nltk
import nltk.tokenize
import collections

subgenres = ('Comédie', 'Tragédie', 'Tragi-comédie') # Three subgenres, Comedy, Tragedy or Trag
plays, titles, genres, authors, dates = [], [], [],[], [] # Initialize empty lists for recursio

for file in os.scandir('theatre-classique'): # For loop through files
    if not file.name.endswith('.xml'): # If the file is not an .xml
       continue # Do nothing and go to next iteration
    tree   = lxml.etree.parse(file.path) # Parse file
    genre  = tree.find('//genre') # Find genre
    title  = tree.find('//title') # Find title
    author = tree.find('//author') # Find author
    date   = tree.find('//date') # Find date
    if genre is not None and genre.text in subgenres: # Parse only plays for which we know the
        lines = []
        for line in tree.xpath('//l|//p'): # The actual play text in these files is matched by
            lines.append(' '.join(line.itertext()))
        text = '\n'.join(lines) # Generate the play
        plays.append(text) # Append the play
        genres.append(genre.text) # Append the genre
        titles.append(title.text) # Append the title
        if author is not None: # There can be missing authors to handle
            authors.append(author.text)
        else:
            authors.append('') # We put an empty string
        if date is not None: # There can be missing dates to handle
            dates.append(date.text)
        else:
            dates.append('') # We put an empty string

print (len(plays), len(genres), len(titles), len(authors), len(dates)) # Should be same size!
```

```
498 498 498 498 498
```

In [1]:
```

```

In [2]:
```
import re # RegExp library
import nltk # Python library for NLP
```

```
punctuation_rule = re.compile(r'[^\w\s]+$') # RegExp that matches punctuations that occur one c

def is_punctuation(string):
    """
    Check if STRING is a punctuation marker or a sequence of
    punctuation markers.
    """
    return punctuation_rule.match(string) is not None # Return punctuation if present

def preprocess_text(text, language='french', lowercase=True):
    """
    Preprocess input text. All to lowercase, sub some common
    French language patterns.
    """
    if lowercase:
        text = text.lower() # All words to lowercase

    if language == 'french': # Preprocess common patterns for French language
        text = re.sub("-", " ", text)
        text = re.sub("l'", "le ", text)
        text = re.sub("d'", "de ", text)
        text = re.sub("c'", "ce ", text)
        text = re.sub("j'", "je ", text)
        text = re.sub("m'", "me ", text)
        text = re.sub("qu'", "que ", text)
        text = re.sub("'", " ' ", text)
        text = re.sub("quelqu'", "quelque ", text)
        text = re.sub("aujourd'hui", "aujourdhui", text)

    tokens = nltk.tokenize.word_tokenize(text, language=language) # Tokenize specifying the lar
    tokens = [token for token in tokens if not is_punctuation(token)] # Exclude punctuations
    return tokens
```

We can finally tokenize our lines as it follows.

In [3]:
```
plays_token = [preprocess_text(play, 'french') for play in plays] # Tokenize every play
```

In [4]:
```
#plays_token
```

These computation let us preprocess the original text and generate a tokenized corpus. Now we can extract from it a vocabulary with a minimum and maximum frequency count.

In [5]:
```
import collections # Library to simplify tallies

def extract_vocabulary(tokenized_corpus, min_count=1, max_count=float('inf')):
    """
    Extract vocabulary from input tokenized text.
    Min frequency count of a vocabulary item is set to 1 and max to infinite.
    """
    vocab = collections.Counter() # Create a container object for rapid tallies
    for document in tokenized_corpus:
        vocab.update(document) # Update for each play
    vocab = {word for word, count in vocab.items()
             if min_count <= count <= max_count} # Append only if the word frequency is betweer
    return sorted(vocab) # Return a list alphabetically ordered of unique words in the corpus

vocabulary = extract_vocabulary(plays_token) # Build the vocabulary
len(vocabulary)
```

Out[5]:
```
62967
```

In [6]:
```
#vocabulary
```

Finally, to represent each play with a vector of term frequencies, we create a document-term matrix (DTM). In this representation, each row is a play in our corpus and each column a unique word with the respective frequency count (tf). The words are ordered as they appear in the play.

In [7]:
```
def corpus2dtm(tokenized_corpus, vocab):
    """
    Custom function to transform a tokenized corpus into a document-term matrix.
    """
    dtm = []
    for document in tokenized_corpus: # For each play
        document_counts = collections.Counter(document) # Get counters
        row = [document_counts[word] for word in vocab] # Count tf for each word in the vocabul
        dtm.append(row) # Append row
    return dtm


document_term_matrix = np.array(corpus2dtm(plays_token, vocabulary)) # Build the DTM
print(f'Document-term matrix with {document_term_matrix.shape[0]} documents and {document_term_
```

```
Document-term matrix with 498 documents and 62967 words.
```

In [11]:
```
#genres
```

In [9]:
```
genres_as_list = np.array(genres) # List to array, for computations

tragedy_profile = document_term_matrix[genres_as_list == 'Tragédie'].mean(axis=0)
comedy_profile = document_term_matrix[genres_as_list == 'Comédie'].mean(axis=0)
tragicomedy_profile = document_term_matrix[genres_as_list == 'Tragi-comédie'].mean(axis=0)

print(tragedy_profile.shape, comedy_profile.shape, tragicomedy_profile.shape) # Single vectors
```

```
(62967,) (62967,) (62967,)
```

In [9]:
```
```

Q2: Which are the three plays for each text genre (or group) that are the "closest" to the profile?

In [10]:
```
# Compute the Euclidan distance
def euclidean_distance(a, b):
    return np.sqrt(np.sum((a - b) ** 2))

tc = euclidean_distance(tragedy_profile, comedy_profile)
print(f'tragédies - comédies:       {tc:.2f}')
ttc = euclidean_distance(tragedy_profile, tragicomedy_profile)
print(f'tragédies - tragi-comédies: {ttc:.2f}')
ctc = euclidean_distance(comedy_profile, tragicomedy_profile)
print(f' comédies - tragi-comédies: {ctc:.2f}')

trag_distances = []
for row in document_term_matrix[genres_as_list == 'Tragédie'].mean(axis=0):
    trag_distances.append(tc)
    tc = euclidean_distance(tragedy_profile, row)
print("Min distance is: " + str(min(trag_distances)) + " which is at index: " +
      str(trag_distances.index(min(trag_distances))) + " for tragedies.")

com_distances = []
for row in document_term_matrix[genres_as_list == 'Comédie'].mean(axis=0):
    ttc = euclidean_distance(comedy_profile, row)
    com_distances.append(ttc)
print("Min distance is: " + str(min(com_distances)) + " which is at index: " +
      str(com_distances.index(min(com_distances))) + " for comedies.")

trc_distances = []
```

```
for row in document_term_matrix[genres_as_list == 'Tragi-comédie'].mean(axis=0):
    ctc = euclidean_distance(tragicomedy_profile, row)
    trc_distances.append(ctc)
print("Min distance is: " + str(min(trc_distances)) + " which is at index: " +
      str(trc_distances.index(min(trc_distances))) + " for tragi-comedie.")
```

```
tragédies - comédies:       447.90
tragédies - tragi-comédies: 301.75
 comédies - tragi-comédies: 656.17
Min distance is: 447.89800112267636 which is at index: 0 for tragedies.
Min distance is: 929.8192306271798 which is at index: 4126 for comedies.
Min distance is: 1533.573447893221 which is at index: 738 for tragi-comedie.
```

Q3: Usually, we generate a profile by averaging over all term frequencies of plays belonging to a certain group. Do you know another way to generate a profile from a set of documents (or vectors)?

As seen in the lecture, we could use the keyness of the terms or the rTF (relative term frequency).

Q4: Do you think that the profile must include all words appearing at least once in a play of the group? If no, how can we select a subset of the terms that must appear in a profile? Justify your choice..

We should definitely only have a subset of words; I vouch for a.) stemming the words to their base forms and b.) removing stop-words. The former gives clearer profiels where certain terms can occur more often, while the latter removes the "noise".

All plays use the most popular words, but those are not what characterize a play or a document. Those, by combining those two, clearer profiles can be made.