Week7: Digital Humanities - Vector Space Model2

In [1]:
```
import os
import lxml.etree
import tarfile
import collections
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import re
import nltk
import nltk.tokenize
import math
import random
```

In [2]:
```
# tf = tarfile.open('theatre-classique.tar.gz','r')
# tf.extractall('data')
```

In [3]:
```
subgenres = ('Comédie', 'Tragédie', 'Tragi-comédie')
#print(subgenres)
plays, titles, genres = [], [], []
authors, years = [],[]
```

In [4]:
```
for fn in os.scandir('data/theatre-classique'):
    # Only include XML files
    if not fn.name.endswith('.xml'):
        continue
    tree   = lxml.etree.parse(fn.path)
    genre  = tree.find('//genre')
    title  = tree.find('//title')
    author = tree.find('//author')
    year   = tree.find('//date')
    if genre is not None and genre.text in subgenres:
        lines = []
        for line in tree.xpath('//l|//p'):
            lines.append(' '.join(line.itertext()))
        text = '\n'.join(lines)
        plays.append(text)
        genres.append(genre.text)
        titles.append(title.text)
        authors.append(author.text)
        if year is not None:
            years.append(year.text)

plays = np.array(plays)
genres = np.array(genres)
titles = np.array(titles)
authors = np.array(authors)
years = np.array(years)
```

In [5]:
```
print(len(plays), len(genres), len(titles), len(authors), len(years))
```

```
498 498 498 498 208
```

In [6]:
```
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Veda\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[6]:
```
True
```

In [7]:
```
def is_punct(string):
    """Check if STRING is a punctuation marker or a sequence of
       punctuation markers.
    """
    return PUNCT_RE.match(string) is not None

PUNCT_RE = re.compile(r'[^\w\s]+$')
```

In [8]:
```
def preprocess_text(text, language='French', lowercase=True):
    if lowercase:
        text = text.lower()
    if (language == 'French'):
        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)
    tokens = [token for token in tokens if not is_punct(token)]
    return tokens
```

In [9]:
```
plays_tok = [preprocess_text(play, 'French') for play in plays]
```

In [10]:
```
def extract_vocabulary(tokenized_corpus, min_count=1, max_count=float('inf')):
    vocabulary = collections.Counter()
    for document in tokenized_corpus:
        vocabulary.update(document)
    vocabulary = {word for word, count in vocabulary.items()
                  if count >= min_count and count <= max_count}
    return sorted(vocabulary)

vocabulary = extract_vocabulary(plays_tok)  # , min_count=2
print("Length of vocabulary: ",len(vocabulary))
```

```
Length of vocabulary:  62967
```

In [11]:
```
def corpus2dtm(tokenized_corpus, vocabulary):
    document_term_matrix = []
    for document in tokenized_corpus:
```

```
        document_counts = collections.Counter(document)
        row = [document_counts[word] for word in vocabulary]
        document_term_matrix.append(row)
    return np.array(document_term_matrix)
```

In [12]:

```
document_term_matrix = np.array(corpus2dtm(plays_tok, vocabulary))
print(f"document-term matrix with "
      f"|D| = {document_term_matrix.shape[0]} documents and "
      f"|V| = {document_term_matrix.shape[1]} words.")
```

```
document-term matrix with |D| = 498 documents and |V| = 62967 words.
```

In [13]:

```
print("Converted doc into vectors :\n",document_term_matrix)
print("\nLength of matrix: \n", len(document_term_matrix))
print("\nSize of matrix: \n", document_term_matrix.shape)
```

```
Converted doc into vectors :
 [[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]

Length of matrix:
 498

Size of matrix:
 (498, 62967)
```

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).

In [14]:

```
tr_means = document_term_matrix[np.array(genres) == 'Tragédie'].mean(axis=0)
co_means = document_term_matrix[genres == 'Comédie'].mean(axis=0)
tc_means = document_term_matrix[genres == 'Tragi-comédie'].mean(axis=0)

print(tr_means.shape, co_means.shape, tc_means.shape)
```

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

Q2. Which are the three plays for each text genre (or group) that are the “closest” to the profile?

In [15]:

```
def vector_len(v):
    """Compute the length (or norm) of a vector."""
    return (np.sqrt(np.sum(v ** 2)) )
```

In [16]:

```
# Cosine distance
def cosine_distance(a, b):
    return ( 1 - np.dot(a, b) / (vector_len(a) * vector_len(b)) )
```

In [17]:
```
tragedy_comedy = cosine_distance(tr_means, co_means)
tragedy_tragedyComedy = cosine_distance(tr_means, tc_means)
tragedyComedy_comedy = cosine_distance(co_means, tc_means)
print(f'tragédies - comédies:       {tragedy_comedy:.2f}')
print(f'tragédies - tragi-comédies: {tragedy_tragedyComedy:.2f}')
print(f' comédies - tragi-comédies: {tragedyComedy_comedy:.2f}')
```

```
tragédies - comédies:       0.03
tragédies - tragi-comédies: 0.01
 comédies - tragi-comédies: 0.02
```

Finding the distance between the profile and each play in the given category:

In [18]:
```
t_dists, c_dists, tc_dists = [], [], []

for aPlay in document_term_matrix[genres == 'Comédie']:
    c_dists.append(cosine_distance(aPlay, co_means))

for aPlay in document_term_matrix[genres == 'Tragédie']:
    t_dists.append(cosine_distance(aPlay, tr_means))

for aPlay in document_term_matrix[genres == 'Tragi-comédie']:
    tc_dists.append(cosine_distance(aPlay, tc_means))
```

In [19]:
```
print("Mean distance to comédie vector:{:.3f}, and Standard deviation:{:.4f}".format(np
print("Mean distance to Tragédie vector:{:.3f}, and Standard deviation:{:.4f}".format(n
print("Mean distance to Tragi-comédie vector:{:.3f},and Standard deviation:{:.4f}".form
```

```
Mean distance to comédie vector:0.040, and Standard deviation:0.0206
Mean distance to Tragédie vector:0.030, and Standard deviation:0.0144
Mean distance to Tragi-comédie vector:0.024,and Standard deviation:0.0092
```

From these arrays, we can sort the distances to find the plays closed to the category vector

In [20]:
```
# top 3 titles of genre comédie which are close to category vector

c_dists = np.array(c_dists)
top_three_c = c_dists.argsort()[:3]
c_titles = np.array(titles)[genres == 'Comédie']
print('\n'.join(c_titles[top_three_c]))
```

```
L'ÉCOLE DES FEMMES, COMÉDIE.
LES MÉNECHMES, ou LES JUMEAUX, COMÉDIE
LA COMÉDIE SANS TITRE, COMÉDIE.
```

In [21]:
```
# top 3 titles of genre Tragédie which are close to category vector

t_dists = np.array(t_dists)
top_three_t = t_dists.argsort()[:3]
t_titles = np.array(titles)[genres == 'Tragédie']
print('\n'.join(t_titles[top_three_t]))
```

```
IRÈNE, TRAGÉDIE
MARIAMNE, TRAGÉDIE EN CINQ ACTES.
GUSTAVE WASA, TRAGÉDIE
```

In [22]:
```
# top 3 titles of genre Tragi-comédie which are close to category vector

tc_dists = np.array(tc_dists)
top_three_tc = tc_dists.argsort()[:3]
tc_titles = np.array(titles)[genres == 'Tragi-comédie']
print('\n'.join(tc_titles[top_three_tc]))
```

```
EURIMÉDON OU L'ILLUSTRE PIRATE. TRAGI-COMÉDIE.
LA BRADAMANTE, TRAGI-COMÉDIE.
LE PRINCE DÉGUISÉ, TRAGI-COMÉDIE
```

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)?

Insteate of generating a profile by averaging over all term frequencies of plays belonging to a certain group, we can generate a profile by using the play which is most typical to the category and we can observed the similar plays.

For ex: Tragedy genre I obtainbelow plays which are almost same as while considering all term frequencies

1. IRÈNE, TRAGÉDIE
2. MARIAMNE, TRAGÉDIE EN CINQ ACTES.
3. SOPHONISBE , TRAGÉDIE

In [59]:
```
top_comedy = np.where(titles == "L'ÉCOLE DES FEMMES, COMÉDIE.")[0]
comedy_play = document_term_matrix[top_comedy]
print(comedy_play.shape)

top_tragedy = np.where(titles == "IRÈNE, TRAGÉDIE")[0]
tragedy_play = document_term_matrix[top_tragedy]
print(tragedy_play.shape)

top_tragedy_comedy = np.where(titles == "EURIMÉDON OU L'ILLUSTRE PIRATE. TRAGI-COMÉDIE.
tragedy_comedy_play = document_term_matrix[top_tragedy_comedy]

t_dists_new, c_dists_new, tc_dists_new = [], [], []

for aPlay in document_term_matrix[genres == 'Comédie']:
    c_dists_new.append(cosine_distance(comedy_play, aPlay))
    #print(aPlay.shape,comedy_play.shape)

print(len(c_dists_new))

#c_play_transport = np.transpose(comedy_play_transport)
for aPlay in document_term_matrix[genres == 'Tragédie']:
    t_dists_new.append(cosine_distance(tragedy_play, aPlay))

for aPlay in document_term_matrix[genres == 'Tragi-comédie']:
    tc_dists_new.append(cosine_distance(tragedy_comedy_play, aPlay))
```

```
(1, 62967)
(1, 62967)
```

```
310
```

In [58]:
```
c_dists_new = np.array(c_dists_new).squeeze()
top_three_c_new = c_dists_new.argsort()[:3]
print(top_three_c_new, c_dists_new.mean())
c_titles_new = np.array(titles)
c_titles_new = c_titles_new[genres == 'Comédie']
print(c_dists_new.shape)

top_three_comedy_titles  = c_titles_new[top_three_c_new]
print('\n'.join([str(v) for v in top_three_comedy_titles]))
```

```
[192 193 202] 0.0514521395612508
(310,)
L'ÉCOLE DES FEMMES, COMÉDIE.
L'ÉCOLE DES MARIS, COMÉDIE
LE MISANTHROPE ou L'ATRABILAIRE AMOUREUX, COMÉDIE
```

In [64]:
```
t_dists_new = np.array(t_dists_new).squeeze()
top_three_t_new = t_dists_new.argsort()[:3]
print(top_three_t_new, t_dists_new.mean())

t_titles_new = np.array(titles)
t_titles_new = t_titles_new[genres == 'Tragédie']
#print(t_dists_new.shape)

top_three_tcomedy_titles  = t_titles_new[top_three_t_new]
print('\n'.join([str(v) for v in top_three_tcomedy_titles]))
```

```
[138 107 117] 0.04151532612412573
IRÈNE, TRAGÉDIE
MARIAMNE, TRAGÉDIE EN CINQ ACTES.
SOPHONISBE , TRAGÉDIE
```

In [66]:
```
tc_dists_new = np.array(tc_dists_new).squeeze()
top_three_tc_new = tc_dists_new.argsort()[:3]
print(top_three_tc_new, tc_dists_new.mean())
tc_titles_new = np.array(titles)
tc_titles_new = tc_titles_new[genres == 'Tragi-comédie']
print(tc_dists_new.shape)

#top_three_tragi-comedy_titles  = tc_titles_new[top_three_tc_new]
print('\n'.join([str(v) for v in tc_titles_new[top_three_tc_new]]))
```

```
[ 0 22 23] 0.03633557480762004
(38,)
EURIMÉDON OU L'ILLUSTRE PIRATE. TRAGI-COMÉDIE.
AGESILAN de COLCHOS, TRAGI-COMÉDIE
AMÉLIE, TRAGI-COMÉDIE
```

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.

selecting the subset of each genre of atleast 50% of plays

Result:

No,No need to include all words in a profile but we can also obtain almost same result with the subset, Here, I compute with the subset of 50% terms. I observe slight reduction in the deviation of the comedie plays from .206 to .182 using the reduced document term matrix. We can examine the top 3 plays for each category remains almost the same compared to the document term matrix, with all the terms considered.

Genre Comedy top 3 plays WITHOUT subset :

L'ÉCOLE DES FEMMES, COMÉDIE.(1) LES MÉNECHMES, ou LES JUMEAUX, COMÉDIE(2) LA COMÉDIE SANS TITRE, COMÉDIE.(3)

Genre Comedy top 3 plays WITH subset :

LES MÉNECHMES, ou LES JUMEAUX, COMÉDIE(1) L'ÉCOLE DES FEMMES, COMÉDIE.(2) LA COMÉDIE SANS TITRE, COMÉDIE.(3)

In [25]:
```
dtm_comedy = pd.DataFrame(document_term_matrix[genres == 'Comédie'])
dtm_comedy = np.array(dtm_comedy.loc[:,dtm_comedy.eq(0).mean().le(.5)])

dtm_tragedy = pd.DataFrame(document_term_matrix[genres == 'Tragédie'])
dtm_tragedy = np.array(dtm_tragedy.loc[:,dtm_tragedy.eq(0).mean().le(.5)])

dtm_tragedy_comedy = pd.DataFrame(document_term_matrix[genres == 'Tragi-comédie'])
dtm_tragedy_comedy = np.array(dtm_tragedy_comedy.loc[:,dtm_tragedy_comedy.eq(0).mean().
```

In [26]:
```
print("dtm_comedy shape:{},\ndtm_tragedy shape:{},\ndtm_tragedy_comedy shape:{}".format
```

```
dtm_comedy shape:(310, 776),
dtm_tragedy shape:(150, 1403),
dtm_tragedy_comedy shape:(38, 1597)
```

In [27]:
```
co_mean_subset = dtm_comedy.mean(axis=0)
tr_mean_subset = dtm_tragedy.mean(axis=0)
tc_mean_subset = dtm_tragedy_comedy.mean(axis=0)
```

In [28]:
```
t_dists_subset, c_dists_subset, tc_dists_subset = [], [], []

for aPlay in dtm_comedy:
    c_dists_subset.append(cosine_distance(aPlay, co_mean_subset))

for aPlay in dtm_tragedy:  #document_term_matrix[genres == 'Tragédie']
    t_dists_subset.append(cosine_distance(aPlay, tr_mean_subset))

for aPlay in dtm_tragedy_comedy:  #document_term_matrix[genres == 'Tragédie']
    tc_dists_subset.append(cosine_distance(aPlay, tc_mean_subset))
```

In [29]:
```
print("Mean distance to subset of comédie vector:{:.3f}, and Standard deviation:{:.4f}"
print("Mean distance to comédie vector:{:.3f}, and Standard deviation:{:.4f}\n".format(
```

```
print("Mean distance to subset of Tragédie vector:{:.3f}, and Standard deviation:{:.4f}
print("Mean distance to Tragédie vector:{:.3f}, and Standard deviation:{:.4f}\n".format

print("Mean distance to subset of Tragi-comédie vector:{:.3f},and Standard deviation:{:
print("Mean distance to Tragi-comédie vector:{:.3f},and Standard deviation:{:.4f}".form
```

```
Mean distance to subset of comédie vector:0.035, and Standard deviation:0.0182
Mean distance to comédie vector:0.040, and Standard deviation:0.0206

Mean distance to subset of Tragédie vector:0.027, and Standard deviation:0.0125
Mean distance to Tragédie vector:0.030, and Standard deviation:0.0144

Mean distance to subset of Tragi-comédie vector:0.022,and Standard deviation:0.0089
Mean distance to Tragi-comédie vector:0.024,and Standard deviation:0.0092
```

In [30]:

```
c_dists_subset = np.array(c_dists_subset)
top_three_subset_c = c_dists_subset.argsort()[:3]
c_titles_subset = np.array(titles)[genres == 'Comédie']
print('\n'.join(c_titles_subset[top_three_subset_c]))
```

```
LES MÉNECHMES, ou LES JUMEAUX, COMÉDIE
L'ÉCOLE DES FEMMES, COMÉDIE.
LA COMÉDIE SANS TITRE, COMÉDIE.
```

In [31]:

```
t_dists_subset = np.array(t_dists_subset)
top_three_subset_t = t_dists_subset.argsort()[:3]
t_titles_subset = np.array(titles)[genres == 'Tragédie']
print('\n'.join(t_titles_subset[top_three_subset_t]))
```

```
IRÈNE, TRAGÉDIE
MARIAMNE, TRAGÉDIE EN CINQ ACTES.
LE COMTE DE WARWIK, TRAGÉDIE.
```

In [32]:

```
tc_dists_subset = np.array(tc_dists_subset)
top_three_subset_tc = tc_dists_subset.argsort()[:3]
tc_titles_subset = np.array(titles)[genres == 'Tragi-comédie']
print('\n'.join(tc_titles_subset[top_three_subset_tc]))
```

```
EURIMÉDON OU L'ILLUSTRE PIRATE. TRAGI-COMÉDIE.
LA BRADAMANTE, TRAGI-COMÉDIE.
DOM QUICHOTTE DE LA MANCHE, COMÉDIE.
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: