Solutions

November 8, 2022

1 Preliminaries

We can load the "french-theater" folder from ILIAS as we did in the second exercise series, and iteratively scrape and parse the content of the .xml files.

[1]:
```
import os
import lxml.etree

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

for file in os.scandir('./french-theater'): # 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 genre
        lines = []
        for line in tree.xpath('//l|//p'): # The actual play text in these␣
↪files is matched by tags p and l
            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
```

2 Question 1: Represent each play by a vector with only the tf component. You can apply some preprocessing before generating this vector representation.

We can define a custom function to preprocess the original play text and latter tokenize each string.

[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 or more times

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 language
    tokens = [token for token in tokens if not is_punctuation(token)] # Exclude␣
 ↪punctuations
    return tokens
```

We can finally tokenize our lines as it follows.

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

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.

[4]:
```
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 between the boundaries
    return sorted(vocab) # Return a list alphabetically ordered of unique words␣
 ↪in the corpus

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

[4]: 63004

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.

[5]:
```
import numpy as np

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 vocabulary
        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_matrix.shape[1]} words.')
```

```
Document-term matrix with 498 documents and 63004 words.
```

3 Question 2: For each genre, it is possible to 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).

We can surely generate a profile (i.e. a "typical" representation of a text) for each genre. A simple strategy that we can follow is just to generate a vector of average frequencies across the row axis.

[6]:
```
genres = np.array(genres) # List to array, for computations

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

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

```
(63004,) (63004,) (63004,)
```

4 Question 3: How many terms with a weight strictly larger than 0 do you have in each text genre profile?

The weight of each term is just its occurrence frequency in the document (*tf*). We can inspect for $tf > 0$ ad it follows.

[7]:
```
print({'tf > 0 (Tragedy)': len(tragedy_profile[tragedy_profile > 0]),
       'tf > 0 (Comedy)': len(comedy_profile[comedy_profile > 0]),
       'tf > 0 (Tragicomedy)': len(tragicomedy_profile[tragicomedy_profile >␣
 ↪0])})
```

```
{'tf > 0 (Tragedy)': 32402, 'tf > 0 (Comedy)': 50268, 'tf > 0 (Tragicomedy)':
17960}
```

5 Question 4: Select randomly 10 plays for each text genre. Represent each play by a vector.

First, we retrieve a set of random indexes going from 0 to the genre size.

[8]:
```
np.random.seed(1234) # Set seed for reproducibility

idxs_tragedy = np.random.randint(low=0, high=len(genres[genres=='Tragédie']),␣
 ↪size=10)
idxs_comedy = np.random.randint(low=0, high=len(genres[genres=='Comédie']),␣
 ↪size=10)
idxs_tragicomedy = np.random.randint(low=0,␣
 ↪high=len(genres[genres=='Tragi-comédie']), size=10)
```

We can then represent each play by a vector from the original DTM matrix.

[9]:
```
tragedy_plays = document_term_matrix[genres == 'Tragédie'][idxs_tragedy]
comedy_plays = document_term_matrix[genres == 'Comédie'][idxs_comedy]
tragicomedy_plays = document_term_matrix[genres ==␣
 ↪'Tragi-comédie'][idxs_tragicomedy]
```

6 Question 5: For each text genre and play, how many terms with a weight strictly larger than 0 do you have in the vector?

[10]:
```
print({'tf > 0 (Tragedy)': len(tragedy_plays[tragedy_plays > 0]),
       'tf > 0 (Comedy)': len(comedy_plays[comedy_plays > 0]),
       'tf > 0 (Tragicomedy)': len(tragicomedy_plays[tragicomedy_plays > 0])})
```

```
{'tf > 0 (Tragedy)': 26260, 'tf > 0 (Comedy)': 19369, 'tf > 0 (Tragicomedy)':
26946}
```

7 Question 6: For each text genre and play, how many terms with a weight strictly equal to 1 do you have in the vector?

[11]:
```
print({'tf == 1 (Tragedy)': len(tragedy_plays[tragedy_plays == 1]),
       'tf == 1 (Comedy)': len(comedy_plays[comedy_plays == 1]),
       'tf == 1 (Tragicomedy)': len(tragicomedy_plays[tragicomedy_plays == 1])})
```

```
{'tf == 1 (Tragedy)': 14692, 'tf == 1 (Comedy)': 11470, 'tf == 1 (Tragicomedy)':
14346}
```