DataStat840Exer4

October 24, 2022

1 DATA.STAT.840 Statistical Methods for Text Data Analysis

Exercises for Lecture 4: Document clustering, and introduction to N-grams Daniel Kusnetsoff

[1]:
```
import requests
import bs4
import re
import nltk
import fnmatch

root = '/Users/danie/'
base_url = 'https://www.gutenberg.org'

#%%
gutenberg_response_html = requests.get(base_url + '/browse/scores/top')
gutenberg_parsed = bs4.BeautifulSoup(gutenberg_response_html.content, 'html.
↪parser')
location = gutenberg_parsed.find(id='books-last30')
book_listing_tag = location.next_sibling.next_sibling
```

[2]:
```
#%%
def get_bookdata(book_listing, k):
    data = []
    for a_tag in book_listing.find_all('a')[:k]:
        book_name = re.match(r'(.*)(\(\d+\))', a_tag.text)
        book_name = book_name.group(1).strip()

        book_id = re.match(r'/ebooks/(\d+)', a_tag.get('href'))
        book_id = book_id.group(1)

        book_url = base_url + '/' + book_id + '/'

        data.append({"id":book_id, "name":book_name, "url":book_url, "fname":
↪''})

    return data
```

[3]:
```python
def get_bookdata_byname(book_listing, name):
    data = []
    for a_tag in book_listing.find_all('a'):
        book_name = re.match(r'(.*)(\(\d+\))', a_tag.text)
        book_name = book_name.group(1).strip()

        if name not in book_name: continue

        book_id = re.match(r'/ebooks/(\d+)', a_tag.get('href'))
        book_id = book_id.group(1)

        book_url = base_url + '/' + book_id + '/'

        data.append({"id":book_id, "name":book_name, "url":book_url, "fname":
↪''})
        break

    return data
```

[4]:
```python
def complete_urls(bookdata):
    for i in range(len(bookdata)):
        print("Searching txt-file for book with id {}".
↪format(bookdata[i]["id"]))
        indexurl = bookdata[i]["url"]

        bookindex_html = requests.get(indexurl)
        bookindex_parsed =  bs4.BeautifulSoup(bookindex_html.content,'html.
↪parser')
        bookindex_links = bookindex_parsed.find_all('a')
        bookindex_hrefs = [bil['href'] for bil in bookindex_links]

        book_filenames = [ bih for bih in bookindex_hrefs if fnmatch.
↪fnmatch(bih, '*.txt') ] #.*.txt

        bookdata[i]["url"] += book_filenames[0]
        bookdata[i]["fname"] = book_filenames[0]

    return
```

[5]:
```python
def download_books(bookdata):
    for data in bookdata:
        print("Downloading book: {}".format(data["name"]))
        print("From source: {}".format(data["url"]))
        response = requests.get(data["url"], allow_redirects=True)
        #txtfiles already made
        with open(root + 'txtfiles/' + data["fname"], 'wb') as f:
            f.write(response.content)
```

```
#%%
book_name = 'The Wonderful Wizard of Oz'
bookdata = get_bookdata_byname(book_listing_tag, book_name)
complete_urls(bookdata)
download_books(bookdata)
```

```
Searching txt-file for book with id 55
Downloading book: The Wonderful Wizard of Oz by L. Frank  Baum
From source: https://www.gutenberg.org/55/55-0.txt
```

[6]:
```
# Download data from locally
def load_data_local(bookdata, fileloc):
    my_text = []
    for data in bookdata:
        try:
            with open(fileloc + data["fname"], 'r') as f:
                my_text.append(f.read())
        except:
            with open(fileloc + data["fname"], 'r', encoding='ISO-8859-1') as f:
                my_text.append(f.read())
        print("Book {} loaded.".format(data['id']))

    return my_text


book_texts = load_data_local(bookdata, root + 'txtfiles/')
```

```
Book 55 loaded.
```

[7]:
```
# Remove the start and end information, so only story text is left

def remove_headers(book_texts):
    start_header = '*** START'
    end_header = '*** END'
    new_texts = []
    for text in book_texts:
        start_loc = text.find(start_header)
        print(start_loc)
        start_loc = text[start_loc:].find('\n') + start_loc
        print(start_loc)
        end_loc = text.find(end_header)
        text = text[start_loc : end_loc]
        new_texts.append(text)

    return new_texts
```

```
book_texts = remove_headers(book_texts)
my_text = book_texts[0]

#%% Get the paragraphs 4.1 b

mytext_paragraphs = paragraphs=re.split('\n[ \n]*\n', my_text)
#paragraphs = paragraphs[1:-1]
```

```
717
788
```

[8]:
```
len(mytext_paragraphs)
```

[8]:
```
1141
```

[9]:
```
paragraph_nltk_texts = []
for k in range(len(mytext_paragraphs)):
    temp_tokenizedtext = nltk.word_tokenize(mytext_paragraphs[k])
    temp_nltktext = nltk.Text(temp_tokenizedtext)
    paragraph_nltk_texts.append(temp_nltktext)
```

[10]:
```
paragraph_nltk_texts[0]
```

[10]:
```
<Text: …>
```

[11]:
```
paragraph_lowercase_texts = []
for k in range(len(paragraph_nltk_texts)):
    temp_lowercase_text = []
    for l in range(len(paragraph_nltk_texts[k])):
        lowercase_word = paragraph_nltk_texts[k][l].lower()
        temp_lowercase_text.append(lowercase_word)
    temp_lowercasetest = nltk.Text(temp_lowercase_text)
    paragraph_lowercase_texts.append(temp_lowercase_text)
```

[12]:
```
#POS
def tagtowordnet(postag):
    wordnettag=-1
    if postag[0]=='N':
        wordnettag='n'
    elif postag[0]=='V':
        wordnettag='v'
    elif postag[0]=='J':
        wordnettag='a'
    elif postag[0]=='R':
        wordnettag='r'
    return(wordnettag)
```

[13]:
```python
# Download wordnet resource if you do not have it already
nltk.download('wordnet')
# Download tagger resource if you do not have it already
nltk.download('averaged_perceptron_tagger')

lemmatizer=nltk.stem.WordNetLemmatizer()
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\danie\AppData\Roaming\nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\danie\AppData\Roaming\nltk_data…
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

[14]:
```python
def lemmatizetext(nltktexttolemmatize):
    # Tag the text with POS tags
    taggedtext=nltk.pos_tag(nltktexttolemmatize)
    # Lemmatize each word text
    lemmatizedtext=[]
    for l in range(len(taggedtext)):
        # Lemmatize a word using the WordNet converted POS tag
        wordtolemmatize=taggedtext[l][0]
        wordnettag=tagtowordnet(taggedtext[l][1])
        if wordnettag!=-1:
            lemmatizedword=lemmatizer.lemmatize(wordtolemmatize,wordnettag)
        else:
            lemmatizedword=wordtolemmatize
        # Store the lemmatized word
        lemmatizedtext.append(lemmatizedword)
    return(lemmatizedtext)


# lemmatization of text
#lemmatized_texts = lemmatizetext(lowercase_texts)
#lemmatized_texts = nltk.Text(lemmatized_texts)
```

[15]:
```python
paragraph_lemmatizedtexts = []
for k in range(len(paragraph_lowercase_texts)):
    lemmatizedtext = lemmatizetext(paragraph_lowercase_texts[k])
    lemmatizedtext = nltk.Text(lemmatizedtext)
    paragraph_lemmatizedtexts.append(lemmatizedtext)
```

[16]:
```python
paragraph_lemmatizedtexts[10]
```

[16]:
```
<Text: have this think in mind , the story…>
```

[17]:
```
import numpy as np
myvocabularies=[]
myindices_in_vocabularies=[]
# Find the vocabulary of each document
for k in range(len(paragraph_lemmatizedtexts)):
    # Get unique words and where they occur
    temptext=paragraph_lemmatizedtexts[k]
    uniqueresults=np.unique(temptext,return_inverse=True)
    uniquewords=uniqueresults[0]
    wordindices=uniqueresults[1]
    # Store the vocabulary and indices of document words in it
    myvocabularies.append(uniquewords)
    myindices_in_vocabularies.append(wordindices)
myvocabularies[0]
```

[17]:
```
array([], dtype=float64)
```

[18]:
```
tempvocabulary=[]
for k in range(len(paragraph_lemmatizedtexts)):
    tempvocabulary.extend(myvocabularies[k])

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

[19]:
```
# Translate previous indices to unified vocabulary.

vocabularystart=0
myindices_in_unifiedvocabulary=[]
for k in range(len(paragraph_lemmatizedtexts)):
    # In order to shift word indices, we must temporarily
    # change their data type to a Numpy array
    tempindices=np.array(myindices_in_vocabularies[k])
    tempindices=tempindices+vocabularystart
    tempindices=wordindices[tempindices]
    myindices_in_unifiedvocabulary.append(tempindices)
    vocabularystart=vocabularystart+len(myvocabularies[k])
```

[20]:
```
unifiedvocabulary_totaloccurrencecounts=np.zeros((len(unifiedvocabulary),1))
unifiedvocabulary_documentcounts=np.zeros((len(unifiedvocabulary),1))
unifiedvocabulary_meancounts=np.zeros((len(unifiedvocabulary),1))
unifiedvocabulary_countvariances=np.zeros((len(unifiedvocabulary),1))
```

[21]:
```
for k in range(len(paragraph_lemmatizedtexts)):
    print(k)
    occurrencecounts=np.zeros((len(unifiedvocabulary),1))
```

```
    for l in range(len(myindices_in_unifiedvocabulary[k])):
        occurrencecounts[myindices_in_unifiedvocabulary[k][l]]= \
            occurrencecounts[myindices_in_unifiedvocabulary[k][l]]+1
    unifiedvocabulary_totaloccurrencecounts= \
        unifiedvocabulary_totaloccurrencecounts+occurrencecounts
    unifiedvocabulary_documentcounts= \
        unifiedvocabulary_documentcounts+(occurrencecounts>0)
```

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
```

[22]:
```
# Mean occurrence counts over documents
unifiedvocabulary_meancounts= \
    unifiedvocabulary_totaloccurrencecounts/len(paragraph_lemmatizedtexts)
```

[23]:
```
#%% Inspect frequent words
# Sort words by largest total (or mean) occurrence count
highest_totaloccurrences_indices=np.argsort(\
    -1*unifiedvocabulary_totaloccurrencecounts,axis=0)
print(np.squeeze(unifiedvocabulary[\
    highest_totaloccurrences_indices[0:100]]))
print(np.squeeze(\
    unifiedvocabulary_totaloccurrencecounts[\
    highest_totaloccurrences_indices[0:100]]))
```

```
['the' ',' 'and' '.' 'be' 'to' 'of' 'a' 'â\x80\x9d' 'in' 'have' 'i' 'he'
 'you' 'her' 'she' 'they' 'it' 'that' 'say' 'dorothy' 'as' 'so' 'for'
 'with' 'not' 'at' 'but' 'all' 'them' 'do' 'scarecrow' 'his' ';' '?' 'me'
 'him' 'my' 'woodman' 'lion' 'come' 'on' 'will' 'oz' 'great' 'when' 'go'
 'make' 'â\x80\x9ci' 'tin' 'ask' 'little' 'witch' 'this' 'from' 'one'
 'could' 'then' 'see' 'there' 'would' 'we' 'if' 'get' 'up' 'out' 'who'
 'head' 'can' 'green' 'their' 'back' 'look' '!' 'no' 'think' 'girl' 'down'
 'know' 'by' 'toto' 'over' 'answer' 'upon' 'shall' 'find' 'give' 'again'
 'good' 'into' 'very' 'now' 'must' 'city' 'wicked' 'where' 'walk' 'after'
 'emerald' 'long']
[2982. 2703. 1600. 1597. 1433. 1110.  840.  801.  698.  481.  476.  452.
  439.  439.  405.  392.  390.  383.  361.  356.  347.  329.  296.  291.
  274.  272.  251.  243.  239.  233.  231.  219.  216.  196.  194.  186.
  179.  178.  175.  175.  169.  157.  156.  151.  150.  148.  148.  144.
  143.  139.  139.  139.  137.  129.  122.  122.  120.  119.  119.  114.
  113.  108.  106.  105.  105.  103.  103.  101.  101.  100.  100.   99.
   97.   97.   94.   94.   94.   93.   93.   90.   90.   87.   86.   85.
   81.   81.   81.   80.   80.   79.   77.   76.   75.   75.   72.   72.
   71.   70.   69.   69.]
```

[24]:
```
nltk.download('stopwords')

#%% Vocabulary pruning
nltkstopwords=nltk.corpus.stopwords.words('english')
pruningdecisions=np.zeros((len(unifiedvocabulary),1))
for k in range(len(unifiedvocabulary)):
    # Rule 1: check the nltk stop word list
    if (unifiedvocabulary[k] in nltkstopwords):
        pruningdecisions[k]=1
    # Rule 2: if the word is in the top 1% of frequent words
    #if (k in highest_totaloccurrences_indices[\
    #    0:int(np.floor(len(unifiedvocabulary)*0.01))]):
    #    pruningdecisions[k]=1
```

```
    # Rule 3: if the word occurs less than 4 times
    if(unifiedvocabulary_totaloccurrencecounts[k] < 4):
        pruningdecisions[k] = 1
    # Rule 4: if the word is too short
    if len(unifiedvocabulary[k])<2:
        pruningdecisions[k]=1
    # Rule 5: if the word is too long
    if len(unifiedvocabulary[k])>20:
        pruningdecisions[k]=1
    # Rule 6: if the word has unwanted characters
    # (here for simplicity only a-z allowed)
    if unifiedvocabulary[k].isalpha()==False:
        pruningdecisions[k]=1
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\danie\AppData\Roaming\nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[25]:
```
#%% Get indices of documents to remaining words
oldtopruned=[]
tempind=-1
for k in range(len(unifiedvocabulary)):
    if pruningdecisions[k]==0:
        tempind=tempind+1
        oldtopruned.append(tempind)
    else:
        oldtopruned.append(-1)
```

[26]:
```
#%% Create pruned texts
paragraph_prunedtexts=[]
myindices_in_prunedvocabulary=[]
for k in range(len(paragraph_lemmatizedtexts)):
    print(k)
    temp_newindices=[]
    temp_newdoc=[]
    for l in range(len(paragraph_lemmatizedtexts[k])):
        temp_oldindex=myindices_in_unifiedvocabulary[k][l]
        temp_newindex=oldtopruned[temp_oldindex]
        if temp_newindex!=-1:
            temp_newindices.append(temp_newindex)
            temp_newdoc.append(unifiedvocabulary[temp_oldindex])
    paragraph_prunedtexts.append(temp_newdoc)
    myindices_in_prunedvocabulary.append(temp_newindices)
```

```
0
1
2
```

1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140

[27]:
```
print('Top 100 word-list after pruning unified vocabulary:\n')
remaining_indices = np.squeeze(np.where(pruningdecisions==0)[0])
remaining_vocabulary = unifiedvocabulary[remaining_indices]
remainingvocabulary_totaloccurrencecounts = unifiedvocabulary_totaloccurrencecounts[remaining_indices]
remaining_highest_totaloccurrences_indices = np.argsort(-1*remainingvocabulary_totaloccurrencecounts, axis=0)
print(np.squeeze(remaining_vocabulary[remaining_highest_totaloccurrences_indices[0:100]]))
```

```
print(np.
↪squeeze(remainingvocabulary_totaloccurrencecounts[remaining_highest_totaloccurrences_indice
↪100]]))
```

```
Top 100 word-list after pruning unified vocabulary:

['say' 'dorothy' 'scarecrow' 'woodman' 'lion' 'come' 'oz' 'great' 'go'
 'make' 'tin' 'ask' 'little' 'witch' 'one' 'could' 'see' 'would' 'get'
 'head' 'green' 'back' 'look' 'think' 'girl' 'know' 'toto' 'answer' 'upon'
 'find' 'shall' 'give' 'good' 'must' 'city' 'wicked' 'walk' 'emerald'
 'long' 'man' 'country' 'room' 'tree' 'away' 'heart' 'like' 'take' 'big'
 'time' 'way' 'carry' 'tell' 'saw' 'people' 'us' 'never' 'eye' 'reply'
 'stand' 'monkey' 'brain' 'live' 'well' 'many' 'chapter' 'day' 'forest'
 'first' 'run' 'road' 'ever' 'friend' 'soon' 'help' 'house' 'around'
 'much' 'keep' 'wish' 'wizard' 'arm' 'cry' 'beast' 'sit' 'thing' 'old'
 'mouse' 'call' 'land' 'beautiful' 'shoe' 'leave' 'air' 'woman' 'seem'
 'put' 'fly' 'quite' 'voice' 'begin']
[356. 347. 219. 175. 175. 169. 151. 150. 148. 144. 139. 139. 139. 137.
 122. 120. 119. 113. 105. 101. 100.  99.  97.  94.  94.  93.  90.  86.
  85.  81.  81.  81.  80.  75.  75.  72.  71.  69.  69.  68.  67.  66.
  65.  65.  65.  65.  64.  63.  62.  60.  59.  58.  58.  57.  56.  55.
  55.  55.  52.  52.  51.  49.  49.  49.  48.  47.  47.  47.  46.  46.
  45.  45.  44.  43.  42.  42.  42.  42.  41.  41.  41.  41.  41.  40.
  40.  39.  38.  38.  38.  38.  37.  36.  36.  35.  35.  35.  35.  35.
  34.  34.]
```

[28]:
```
import scipy
#%% Create TF-IDF vectors
n_docs=len(paragraph_prunedtexts)
n_vocab=len(remaining_vocabulary)
# Matrix of term frequencies
tfmatrix=scipy.sparse.lil_matrix((n_docs,n_vocab))
# Row vector of document frequencies
dfvector=scipy.sparse.lil_matrix((1,n_vocab))
# Loop over documents
for k in range(n_docs):
    # 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(paragraph_prunedtexts[k])):
        # Add current word to term-frequency count and document-count
        currentword=myindices_in_prunedvocabulary[k][l]
        tfmatrix[k,currentword]=tfmatrix[k,currentword]+1
        temp_dfvector[0,currentword]=1
    # Add which words occurred in this document to overall document counts
    dfvector=dfvector+temp_dfvector
```

```
# TF:length-normalized frequency
for i in range(n_docs):
    for j in range(len(tfmatrix.data[i])):
        tfmatrix.data[i][j] = tfmatrix.data[i][j]/len(tfmatrix.data[i])



# smoothed logarithmic idf
idfvector = np.squeeze(np.array(dfvector.todense()))
idfvector = np.log(1 + ((idfvector+1)**-1)*n_docs)

tfidfmatrix = scipy.sparse.lil_matrix((n_docs, n_vocab))
for k in range(n_docs):
    # tf and idf terms
    tfidfmatrix[k,:]=tfmatrix[k,:]*idfvector

# tf-idf matrix
#tfidfmatrix = scipy.sparse.lil_matrix((n_docs, n_vocab))
for k in range(n_docs):
    # find nonzero term frequencies
    tempindices = np.nonzero(tfmatrix[k, :])[1]
    tfterm = np.squeeze(np.array(tfmatrix[k, tempindices].todense()))
    tfidfmatrix[k, tempindices] = tfterm * idfvector[tempindices]
```

```
C:\Users\danie\Anaconda3\lib\site-packages\scipy\sparse\lil.py:512:
FutureWarning: future versions will not create a writeable array from
broadcast_array. Set the writable flag explicitly to avoid this warning.
  if not i.flags.writeable or i.dtype not in (np.int32, np.int64):
C:\Users\danie\Anaconda3\lib\site-packages\scipy\sparse\lil.py:514:
FutureWarning: future versions will not create a writeable array from
broadcast_array. Set the writable flag explicitly to avoid this warning.
  if not j.flags.writeable or j.dtype not in (np.int32, np.int64):
C:\Users\danie\Anaconda3\lib\site-packages\scipy\sparse\lil.py:518:
FutureWarning: future versions will not create a writeable array from
broadcast_array. Set the writable flag explicitly to avoid this warning.
  if not x.flags.writeable:
```

[29]:

```
#%% Use the TF-IDF matrix as data to be clustered
X=tfidfmatrix
# Normalize the documents to unit vector norm
tempnorms=np.squeeze(np.array(np.sum(X.multiply(X),axis=1)))
# If any documents have zero norm, avoid dividing them by zero
tempnorms[tempnorms==0]=1
X=scipy.sparse.diags(tempnorms**-0.5).dot(X)
n_data=np.shape(X)[0]
n_dimensions=np.shape(X)[1]
```

[30]:
```
# Function to initialize the Gaussian mixture model, create component parameters
def initialize_mixturemodel(X,n_components):
    # Create lists of sparse matrices to hold the parameters
    n_dimensions=np.shape(X)[1]
    n_data = np.shape(X)[0]
    mixturemodel_means=scipy.sparse.lil_matrix((n_components,n_dimensions))
    mixturemodel_weights=np.zeros((n_components))
    mixturemodel_covariances=[]
    mixturemodel_inversecovariances=[]
    for k in range(n_components):
        tempcovariance=scipy.sparse.lil_matrix((n_dimensions,n_dimensions))
        mixturemodel_covariances.append(tempcovariance)
        tempinvcovariance=scipy.sparse.lil_matrix((n_dimensions,n_dimensions))
        mixturemodel_inversecovariances.append(tempinvcovariance)
    # Initialize the parameters
    for k in range(n_components):
        mixturemodel_weights[k]=1/n_components
        # Pick a random data point as the initial mean
        tempindex=scipy.stats.randint.rvs(low=0,high=n_data)
        mixturemodel_means[k]=X[tempindex,:].toarray()
        # Initialize the covariance matrix to be spherical
        for l in range(n_dimensions):
            mixturemodel_covariances[k][l,l]=1
            mixturemodel_inversecovariances[k][l,l]=1
    ␣
↪return(mixturemodel_weights,mixturemodel_means,mixturemodel_covariances,mixturemodel_invers
```

[31]:
```
def␣
↪run_estep(X,mixturemodel_means,mixturemodel_covariances,mixturemodel_inversecovariances,mix
↪
    # For each component, compute terms that do not involve data
    meanterms=np.zeros((n_components))
    logdeterminants=np.zeros((n_components))
    logconstantterms=np.zeros((n_components))

    for k in range(n_components):
        # Compute mu_k*inv(Sigma_k)*mu_k
        meanterms[k]=(mixturemodel_means[k,:
↪]*mixturemodel_inversecovariances[k]*mixturemodel_means[k,:].T)[0,0]
        # Compute determinant of Sigma_k. For a diagonal matrix
        # this is just the product of the main diagonal
        logdeterminants[k]=np.sum(np.log(mixturemodel_covariances[k].
↪diagonal(0)))
        # Compute constant term beta_k * 1/(|Sigma_k|^1/2)
        # Omit the (2pi)^d/2 as it cancels out
        logconstantterms[k]=np.log(mixturemodel_weights[k]) - 0.
↪5*logdeterminants[k]
```

```
    print('E-step part2 ')
    # Compute terms that involve distances of data from components
    xnorms=np.zeros((n_data,n_components))
    xtimesmu=np.zeros((n_data,n_components))

    for k in range(n_components):
        #print(k)
        xnorms[:,k]=(X*mixturemodel_inversecovariances[k]*X.T).diagonal(0)
        xtimesmu[:,k]=np.squeeze((X*mixturemodel_inversecovariances[k]*␣
↪mixturemodel_means[k,:].T).toarray())

    xdists=xnorms+np.matlib.repmat(meanterms,n_data,1)-2*xtimesmu
    # Substract maximal term before exponent (cancels out) to maintain␣
↪computational precision
    numeratorterms=logconstantterms-xdists/2
    numeratorterms-=np.matlib.repmat(np.
↪max(numeratorterms,axis=1),n_components,1).T
    numeratorterms=np.exp(numeratorterms)
    mixturemodel_componentmemberships=numeratorterms/np.matlib.repmat(np.
↪sum(numeratorterms,axis=1),n_components,1).T
    return(mixturemodel_componentmemberships)
```

[32]:
```
def run_mstep_sumweights(mixturemodel_componentmemberships):
    # Compute total weight per component
    mixturemodel_weights=np.sum(mixturemodel_componentmemberships,axis=0)
    return(mixturemodel_weights)
```

[33]:
```
def run_mstep_means(X,mixturemodel_componentmemberships,mixturemodel_weights):
    # Update component means
    mixturemodel_means=scipy.sparse.lil_matrix((n_components,n_dimensions))
    for k in range(n_components):
        mixturemodel_means[k,:]=\
            np.sum(scipy.sparse.diags(mixturemodel_componentmemberships[:,k]).
↪dot(X),axis=0)
        mixturemodel_means[k,:]/=mixturemodel_weights[k]
    return(mixturemodel_means)
```

[34]:
```
def␣
↪run_mstep_covariances(X,mixturemodel_componentmemberships,mixturemodel_weights,mixturemodel_
↪
    # Update diagonal component covariance matrices
    n_dimensions=np.shape(X)[1]
    n_components=np.shape(mixturemodel_componentmemberships)[1]
    tempcovariances=np.zeros((n_components,n_dimensions))
    mixturemodel_covariances=[]
```

```
    mixturemodel_inversecovariances=[]

    for k in range(n_components):
        tempcovariances[k,:]= np.sum(scipy.sparse.diags(
            mixturemodel_componentmemberships[:,k]).dot(
            X.multiply(X)),axis=0)-mixturemodel_means[k,:].
↪multiply(mixturemodel_means[k,:])*mixturemodel_weights[k]
        tempcovariances[k,:]/=mixturemodel_weights[k]
        # Convert to sparse matrices
        tempepsilon=1e-10
        # Add a small regularization term
        temp_covariance=scipy.sparse.diags(tempcovariances[k,:]+tempepsilon)
        temp_inversecovariance=scipy.sparse.diags((tempcovariances[k,:
↪]+tempepsilon)**-1)
        mixturemodel_covariances.append(temp_covariance)
        mixturemodel_inversecovariances.append(temp_inversecovariance)
    return(mixturemodel_covariances,mixturemodel_inversecovariances)
```

[35]:
```
def run_mstep_normalizeweights(mixturemodel_weights):
    # Update mixture-component prior probabilities
    mixturemodel_weights/=sum(mixturemodel_weights)
    return(mixturemodel_weights)
```

[36]:
```
#%% Perform the EM algorithm iterations
def perform_emalgorithm(X,n_components,n_emiterations):
    mixturemodel_weights,mixturemodel_means,mixturemodel_covariances,\
    mixturemodel_inversecovariances=initialize_mixturemodel(X,n_components)
    for t in range(n_emiterations):
        # ====== E-step: Compute the component membership
        # probabilities of each data point ======
        print('E-step ' + str(t))
        ␣
↪mixturemodel_componentmemberships=run_estep(X,mixturemodel_means,mixturemodel_covariances,\
        mixturemodel_inversecovariances,mixturemodel_weights)
        # ====== M-step: update component parameters======
        print('M-step ' + str(t))
        print('M-step part1 ' + str(t))
        ␣
↪mixturemodel_weights=run_mstep_sumweights(mixturemodel_componentmemberships)
        print('M-step part2 ' + str(t))
        ␣
↪mixturemodel_means=run_mstep_means(X,mixturemodel_componentmemberships,mixturemodel_weights
        print('M-step part3 ' + str(t))
        ␣
↪mixturemodel_covariances,mixturemodel_inversecovariances=run_mstep_covariances(X,\
```

```
        ␣
↪mixturemodel_componentmemberships,mixturemodel_weights,mixturemodel_means)
        print('M-step part4 ' + str(t))
        mixturemodel_weights=run_mstep_normalizeweights(mixturemodel_weights)
    return(mixturemodel_weights,mixturemodel_means,mixturemodel_covariances,\
mixturemodel_inversecovariances)
# Try out the functions we just defined on the data
n_components=10
n_emiterations=100
mixturemodel_weights,mixturemodel_means,mixturemodel_covariances,\
mixturemodel_inversecovariances =␣
↪perform_emalgorithm(X,n_components,n_emiterations)
```

```
C:\Users\danie\Anaconda3\lib\site-packages\scipy\sparse\lil.py:512:
FutureWarning: future versions will not create a writeable array from
broadcast_array. Set the writable flag explicitly to avoid this warning.
  if not i.flags.writeable or i.dtype not in (np.int32, np.int64):
C:\Users\danie\Anaconda3\lib\site-packages\scipy\sparse\lil.py:514:
FutureWarning: future versions will not create a writeable array from
broadcast_array. Set the writable flag explicitly to avoid this warning.
  if not j.flags.writeable or j.dtype not in (np.int32, np.int64):
```

```
E-step 0
E-step part2
M-step 0
M-step part1 0
M-step part2 0
M-step part3 0
M-step part4 0
E-step 1
E-step part2
M-step 1
M-step part1 1
M-step part2 1
M-step part3 1
M-step part4 1
E-step 2
E-step part2
M-step 2
M-step part1 2
M-step part2 2
M-step part3 2
M-step part4 2
E-step 3
E-step part2
M-step 3
M-step part1 3
M-step part2 3
```

```
M-step part3 3
M-step part4 3
E-step 4
E-step part2
M-step 4
M-step part1 4
M-step part2 4
M-step part3 4
M-step part4 4
E-step 5
E-step part2
M-step 5
M-step part1 5
M-step part2 5
M-step part3 5
M-step part4 5
E-step 6
E-step part2
M-step 6
M-step part1 6
M-step part2 6
M-step part3 6
M-step part4 6
E-step 7
E-step part2
M-step 7
M-step part1 7
M-step part2 7
M-step part3 7
M-step part4 7
E-step 8
E-step part2
M-step 8
M-step part1 8
M-step part2 8
M-step part3 8
M-step part4 8
E-step 9
E-step part2
M-step 9
M-step part1 9
M-step part2 9
M-step part3 9
M-step part4 9
E-step 10
E-step part2
M-step 10
M-step part1 10
```

```
M-step part2 10
M-step part3 10
M-step part4 10
E-step 11
E-step part2
M-step 11
M-step part1 11
M-step part2 11
M-step part3 11
M-step part4 11
E-step 12
E-step part2
M-step 12
M-step part1 12
M-step part2 12
M-step part3 12
M-step part4 12
E-step 13
E-step part2
M-step 13
M-step part1 13
M-step part2 13
M-step part3 13
M-step part4 13
E-step 14
E-step part2
M-step 14
M-step part1 14
M-step part2 14
M-step part3 14
M-step part4 14
E-step 15
E-step part2
M-step 15
M-step part1 15
M-step part2 15
M-step part3 15
M-step part4 15
E-step 16
E-step part2
M-step 16
M-step part1 16
M-step part2 16
M-step part3 16
M-step part4 16
E-step 17
E-step part2
M-step 17
```

```
M-step part1 17
M-step part2 17
M-step part3 17
M-step part4 17
E-step 18
E-step part2
M-step 18
M-step part1 18
M-step part2 18
M-step part3 18
M-step part4 18
E-step 19
E-step part2
M-step 19
M-step part1 19
M-step part2 19
M-step part3 19
M-step part4 19
E-step 20
E-step part2
M-step 20
M-step part1 20
M-step part2 20
M-step part3 20
M-step part4 20
E-step 21
E-step part2
M-step 21
M-step part1 21
M-step part2 21
M-step part3 21
M-step part4 21
E-step 22
E-step part2
M-step 22
M-step part1 22
M-step part2 22
M-step part3 22
M-step part4 22
E-step 23
E-step part2
M-step 23
M-step part1 23
M-step part2 23
M-step part3 23
M-step part4 23
E-step 24
E-step part2
```

```
M-step 24
M-step part1 24
M-step part2 24
M-step part3 24
M-step part4 24
E-step 25
E-step part2
M-step 25
M-step part1 25
M-step part2 25
M-step part3 25
M-step part4 25
E-step 26
E-step part2
M-step 26
M-step part1 26
M-step part2 26
M-step part3 26
M-step part4 26
E-step 27
E-step part2
M-step 27
M-step part1 27
M-step part2 27
M-step part3 27
M-step part4 27
E-step 28
E-step part2
M-step 28
M-step part1 28
M-step part2 28
M-step part3 28
M-step part4 28
E-step 29
E-step part2
M-step 29
M-step part1 29
M-step part2 29
M-step part3 29
M-step part4 29
E-step 30
E-step part2
M-step 30
M-step part1 30
M-step part2 30
M-step part3 30
M-step part4 30
E-step 31
```

```
E-step part2
M-step 31
M-step part1 31
M-step part2 31
M-step part3 31
M-step part4 31
E-step 32
E-step part2
M-step 32
M-step part1 32
M-step part2 32
M-step part3 32
M-step part4 32
E-step 33
E-step part2
M-step 33
M-step part1 33
M-step part2 33
M-step part3 33
M-step part4 33
E-step 34
E-step part2
M-step 34
M-step part1 34
M-step part2 34
M-step part3 34
M-step part4 34
E-step 35
E-step part2
M-step 35
M-step part1 35
M-step part2 35
M-step part3 35
M-step part4 35
E-step 36
E-step part2
M-step 36
M-step part1 36
M-step part2 36
M-step part3 36
M-step part4 36
E-step 37
E-step part2
M-step 37
M-step part1 37
M-step part2 37
M-step part3 37
M-step part4 37
```

```
E-step 38
E-step part2
M-step 38
M-step part1 38
M-step part2 38
M-step part3 38
M-step part4 38
E-step 39
E-step part2
M-step 39
M-step part1 39
M-step part2 39
M-step part3 39
M-step part4 39
E-step 40
E-step part2
M-step 40
M-step part1 40
M-step part2 40
M-step part3 40
M-step part4 40
E-step 41
E-step part2
M-step 41
M-step part1 41
M-step part2 41
M-step part3 41
M-step part4 41
E-step 42
E-step part2
M-step 42
M-step part1 42
M-step part2 42
M-step part3 42
M-step part4 42
E-step 43
E-step part2
M-step 43
M-step part1 43
M-step part2 43
M-step part3 43
M-step part4 43
E-step 44
E-step part2
M-step 44
M-step part1 44
M-step part2 44
M-step part3 44
```

```
M-step part4 44
E-step 45
E-step part2
M-step 45
M-step part1 45
M-step part2 45
M-step part3 45
M-step part4 45
E-step 46
E-step part2
M-step 46
M-step part1 46
M-step part2 46
M-step part3 46
M-step part4 46
E-step 47
E-step part2
M-step 47
M-step part1 47
M-step part2 47
M-step part3 47
M-step part4 47
E-step 48
E-step part2
M-step 48
M-step part1 48
M-step part2 48
M-step part3 48
M-step part4 48
E-step 49
E-step part2
M-step 49
M-step part1 49
M-step part2 49
M-step part3 49
M-step part4 49
E-step 50
E-step part2
M-step 50
M-step part1 50
M-step part2 50
M-step part3 50
M-step part4 50
E-step 51
E-step part2
M-step 51
M-step part1 51
M-step part2 51
```

```
M-step part3 51
M-step part4 51
E-step 52
E-step part2
M-step 52
M-step part1 52
M-step part2 52
M-step part3 52
M-step part4 52
E-step 53
E-step part2
M-step 53
M-step part1 53
M-step part2 53
M-step part3 53
M-step part4 53
E-step 54
E-step part2
M-step 54
M-step part1 54
M-step part2 54
M-step part3 54
M-step part4 54
E-step 55
E-step part2
M-step 55
M-step part1 55
M-step part2 55
M-step part3 55
M-step part4 55
E-step 56
E-step part2
M-step 56
M-step part1 56
M-step part2 56
M-step part3 56
M-step part4 56
E-step 57
E-step part2
M-step 57
M-step part1 57
M-step part2 57
M-step part3 57
M-step part4 57
E-step 58
E-step part2
M-step 58
M-step part1 58
```

```
M-step part2 58
M-step part3 58
M-step part4 58
E-step 59
E-step part2
M-step 59
M-step part1 59
M-step part2 59
M-step part3 59
M-step part4 59
E-step 60
E-step part2
M-step 60
M-step part1 60
M-step part2 60
M-step part3 60
M-step part4 60
E-step 61
E-step part2
M-step 61
M-step part1 61
M-step part2 61
M-step part3 61
M-step part4 61
E-step 62
E-step part2
M-step 62
M-step part1 62
M-step part2 62
M-step part3 62
M-step part4 62
E-step 63
E-step part2
M-step 63
M-step part1 63
M-step part2 63
M-step part3 63
M-step part4 63
E-step 64
E-step part2
M-step 64
M-step part1 64
M-step part2 64
M-step part3 64
M-step part4 64
E-step 65
E-step part2
M-step 65
```

```
M-step part1 65
M-step part2 65
M-step part3 65
M-step part4 65
E-step 66
E-step part2
M-step 66
M-step part1 66
M-step part2 66
M-step part3 66
M-step part4 66
E-step 67
E-step part2
M-step 67
M-step part1 67
M-step part2 67
M-step part3 67
M-step part4 67
E-step 68
E-step part2
M-step 68
M-step part1 68
M-step part2 68
M-step part3 68
M-step part4 68
E-step 69
E-step part2
M-step 69
M-step part1 69
M-step part2 69
M-step part3 69
M-step part4 69
E-step 70
E-step part2
M-step 70
M-step part1 70
M-step part2 70
M-step part3 70
M-step part4 70
E-step 71
E-step part2
M-step 71
M-step part1 71
M-step part2 71
M-step part3 71
M-step part4 71
E-step 72
E-step part2
```

```
M-step 72
M-step part1 72
M-step part2 72
M-step part3 72
M-step part4 72
E-step 73
E-step part2
M-step 73
M-step part1 73
M-step part2 73
M-step part3 73
M-step part4 73
E-step 74
E-step part2
M-step 74
M-step part1 74
M-step part2 74
M-step part3 74
M-step part4 74
E-step 75
E-step part2
M-step 75
M-step part1 75
M-step part2 75
M-step part3 75
M-step part4 75
E-step 76
E-step part2
M-step 76
M-step part1 76
M-step part2 76
M-step part3 76
M-step part4 76
E-step 77
E-step part2
M-step 77
M-step part1 77
M-step part2 77
M-step part3 77
M-step part4 77
E-step 78
E-step part2
M-step 78
M-step part1 78
M-step part2 78
M-step part3 78
M-step part4 78
E-step 79
```

```
E-step part2
M-step 79
M-step part1 79
M-step part2 79
M-step part3 79
M-step part4 79
E-step 80
E-step part2
M-step 80
M-step part1 80
M-step part2 80
M-step part3 80
M-step part4 80
E-step 81
E-step part2
M-step 81
M-step part1 81
M-step part2 81
M-step part3 81
M-step part4 81
E-step 82
E-step part2
M-step 82
M-step part1 82
M-step part2 82
M-step part3 82
M-step part4 82
E-step 83
E-step part2
M-step 83
M-step part1 83
M-step part2 83
M-step part3 83
M-step part4 83
E-step 84
E-step part2
M-step 84
M-step part1 84
M-step part2 84
M-step part3 84
M-step part4 84
E-step 85
E-step part2
M-step 85
M-step part1 85
M-step part2 85
M-step part3 85
M-step part4 85
```

```
E-step 86
E-step part2
M-step 86
M-step part1 86
M-step part2 86
M-step part3 86
M-step part4 86
E-step 87
E-step part2
M-step 87
M-step part1 87
M-step part2 87
M-step part3 87
M-step part4 87
E-step 88
E-step part2
M-step 88
M-step part1 88
M-step part2 88
M-step part3 88
M-step part4 88
E-step 89
E-step part2
M-step 89
M-step part1 89
M-step part2 89
M-step part3 89
M-step part4 89
E-step 90
E-step part2
M-step 90
M-step part1 90
M-step part2 90
M-step part3 90
M-step part4 90
E-step 91
E-step part2
M-step 91
M-step part1 91
M-step part2 91
M-step part3 91
M-step part4 91
E-step 92
E-step part2
M-step 92
M-step part1 92
M-step part2 92
M-step part3 92
```

```
M-step part4 92
E-step 93
E-step part2
M-step 93
M-step part1 93
M-step part2 93
M-step part3 93
M-step part4 93
E-step 94
E-step part2
M-step 94
M-step part1 94
M-step part2 94
M-step part3 94
M-step part4 94
E-step 95
E-step part2
M-step 95
M-step part1 95
M-step part2 95
M-step part3 95
M-step part4 95
E-step 96
E-step part2
M-step 96
M-step part1 96
M-step part2 96
M-step part3 96
M-step part4 96
E-step 97
E-step part2
M-step 97
M-step part1 97
M-step part2 97
M-step part3 97
M-step part4 97
E-step 98
E-step part2
M-step 98
M-step part1 98
M-step part2 98
M-step part3 98
M-step part4 98
E-step 99
E-step part2
M-step 99
M-step part1 99
M-step part2 99
```

```
M-step part3 99
M-step part4 99
```

[37]:
```python
for k in range(n_components):
    print(k)
    highest_dimensionweight_indices=np.argsort(-np.
 ↪squeeze(mixturemodel_means[k,:].toarray()),axis=0)

    print(' '.join(remaining_vocabulary[highest_dimensionweight_indices[1:10]]))
```

```
0
stranger gravely grateful gown street stretch gold strike glass
1
party pass pat pave perch person pick part piece
2
nearly nearer near mouth monster milkmaid mile midst middle
3
place play pleasant please pleased plenty point pole poor
4
often oil open order others ought palace offer part
5
ought paint party pat patch pave perch pick piece
6
mistake tire tip monster morning mouth move tiny munchkin
7
rock gently gaze gayelette gather roof funny rope fruit
8
inquire ruler remark dear spider figure five scare inside
9
inquire indeed immediately imagine spider hung humbug spite spot
```

[38]:
```python
# Version 2 - Get documents closest to component mean, i.e. highest p(d|k).
# ---The computation of distances here is the same as done in the E-step of␣
 ↪EM---
# For each component, compute terms that do not involve data
meanterms=np.zeros((n_components))
logdeterminants=np.zeros((n_components))
logconstantterms=np.zeros((n_components))
for k in range(n_components):
    # Compute mu_k*inv(Sigma_k)*mu_k
    meanterms[k]=(mixturemodel_means[k,:
 ↪]*mixturemodel_inversecovariances[k]*mixturemodel_means[k,:].T)[0,0]

# Compute terms that involve distances of data from components
xnorms=np.zeros((n_data,n_components))
xtimesmu=np.zeros((n_data,n_components))
```

```
for k in range(n_components):
    xnorms[:,k]=(X*mixturemodel_inversecovariances[k]*X.T).diagonal(0)
    xtimesmu[:,k]=np.
 ↪squeeze((X*mixturemodel_inversecovariances[k]*mixturemodel_means[k,:].T).
 ↪toarray())

xdists=xnorms+np.matlib.repmat(meanterms,n_data,1)-2*xtimesmu

for k in range(n_components):
    tempdists=np.array(np.squeeze(xdists[:,k]))
    highest_componentprob_indices=np.argsort(-1*tempdists,axis=0)
    print(k)
    print(highest_componentprob_indices[0:10])
    print(' '.join(paragraph_nltk_texts[highest_componentprob_indices[0]]))
```

```
0
[   0  876  635  633  631  420   11 1140    1    7]

1
[   0  876  635  633  631  420   11 1140    7    1]

2
[   0  876  635  633  631  420   11 1140    7    3]

3
[   0  876  635  633  631  420   11 1140    7    3]

4
[   0  876  635  633  631  420   11 1140    3    7]

5
[   0  876  635  633  631  420   11 1140    7    3]

6
[   0  876  635  633  631  420   11 1140    7    1]

7
[   0  876  635  633  631  420   11 1140    1    7]

8
[   0  876  635  633  631  420   11 1140    3    7]

9
[   0  876  635  633  631  420   11 1140    1    7]
```

g) In every mixture component, the highest membership probability does not match with the current top-10 words in the same mixture component.

The result might be due to that the paragraph in question might be very near the cluster center. This might happen as it is built out of rather generic words that might be common in every cluster.

A few of the paragraphs with the highest membership probability gave the same result. Is this correct.

For some reason, the words dissappeared in the last part of the task, even though they were prior visible.

2 EX 4.2

[39]:
```
#Find out the amout of words in paragraphs
word_count = np.zeros((len(paragraph_lemmatizedtexts), 1))
for k in range(len(paragraph_lemmatizedtexts)):
    counting = len(paragraph_lemmatizedtexts[k])
    word_count[k] = counting
```

[40]:
```
max(word_count)
```

[40]:
```
array([234.])
```

[41]:
```
no_longest_paragraph = np.argsort(-1*word_count, axis=0)[0]
```

[42]:
```
no_longest_paragraph
```

[42]:
```
array([505], dtype=int64)
```

[43]:
```
#The longest paragraph
print(' '.join(paragraph_lemmatizedtexts[505]))
```

```
she leave dorothy alone and go back to the others . these she also lead to room
, and each one of them find himself lodge in a very pleasant part of the palace
. of course this politeness be waste on the scarecrow ; for when he find himself
alone in his room he stand stupidly in one spot , just within the doorway , to
wait till morning . it would not rest him to lie down , and he could not close
his eye ; so he remain all night star at a little spider which be weave its web
in a corner of the room , just as if it be not one of the most wonderful room in
the world . the tin woodman lay down on his bed from force of habit , for he
remember when he be make of flesh ; but not be able to sleep , he pass the night
move his joint up and down to make sure they keep in good work order . the lion
would have prefer a bed of dried leaf in the forest , and do not like be shut up
in a room ; but he have too much sense to let this worry him , so he spring upon
the bed and roll himself up like a cat and purr himself asleep in a minute .
```

[44]:
```
longestpara=(' '.join(paragraph_lemmatizedtexts[505]))
longestpara
```

[44]: 'she leave dorothy alone and go back to the others . these she also lead to room , and each one of them find himself lodge in a very pleasant part of the palace . of course this politeness be waste on the scarecrow ; for when he find himself alone in his room he stand stupidly in one spot , just within the doorway , to wait till morning . it would not rest him to lie down , and he could not close his eye ; so he remain all night star at a little spider which be weave its web in a corner of the room , just as if it be not one of the most wonderful room in the world . the tin woodman lay down on his bed from force of habit , for he remember when he be make of flesh ; but not be able to sleep , he pass the night move his joint up and down to make sure they keep in good work order . the lion would have prefer a bed of dried leaf in the forest , and do not like be shut up in a room ; but he have too much sense to let this worry him , so he spring upon the bed and roll himself up like a cat and purr himself asleep in a minute .'

[45]:
```
longestmyvocabularies=[]
longestmyindices_in_vocabularies=[]
# Find the vocabulary of each document
for k in range(len(longestpara)):
    # Get unique words and where they occur
    temptext=longestpara[k]
    uniqueresults=np.unique(temptext,return_inverse=True)
    uniquewords=uniqueresults[0]
    wordindices=uniqueresults[1]
    # Store the vocabulary and indices of document words in it
    longestmyvocabularies.append(uniquewords)
    longestmyindices_in_vocabularies.append(wordindices)
longestmyvocabularies[0]
```

[45]: array(['s'], dtype='<U1')

[46]:
```
tempvocabulary=[]
for k in range(len(paragraph_lemmatizedtexts)):
    tempvocabulary.extend(myvocabularies[k])

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

[47]:
```
# Translate previous indices to unified vocabulary.

vocabularystart=0
myindices_in_unifiedvocabulary=[]
for k in range(len(paragraph_lemmatizedtexts)):
    # In order to shift word indices, we must temporarily
    # change their data type to a Numpy array
    tempindices=np.array(myindices_in_vocabularies[k])
```

```
    tempindices=tempindices+vocabularystart
    tempindices=wordindices[tempindices]
    myindices_in_unifiedvocabulary.append(tempindices)
    vocabularystart=vocabularystart+len(myvocabularies[k])
```

[48]:
```
unifiedvocabulary_totaloccurrencecounts=np.zeros((len(unifiedvocabulary),1))
unifiedvocabulary_documentcounts=np.zeros((len(unifiedvocabulary),1))
unifiedvocabulary_meancounts=np.zeros((len(unifiedvocabulary),1))
unifiedvocabulary_countvariances=np.zeros((len(unifiedvocabulary),1))
```

[49]:
```
for k in range(len(longestpara)):
    print(k)
    occurrencecounts=np.zeros((len(unifiedvocabulary),1))
    for l in range(len(myindices_in_unifiedvocabulary[k])):
        occurrencecounts[myindices_in_unifiedvocabulary[k][l]]= \
            occurrencecounts[myindices_in_unifiedvocabulary[k][l]]+1
    unifiedvocabulary_totaloccurrencecounts= \
        unifiedvocabulary_totaloccurrencecounts+occurrencecounts
    unifiedvocabulary_documentcounts= \
        unifiedvocabulary_documentcounts+(occurrencecounts>0)
```

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
```

```
1034
1035
```

[50]:
```
# Mean occurrence counts over documents
unifiedvocabulary_meancounts= \
    unifiedvocabulary_totaloccurrencecounts/len(longestpara)
```

[51]:
```
#%% Inspect frequent words
# Sort words by largest total (or mean) occurrence count
highest_totaloccurrences_indices=np.argsort(\
    -1*unifiedvocabulary_totaloccurrencecounts,axis=0)
print(np.squeeze(unifiedvocabulary[\
    highest_totaloccurrences_indices[0:100]]))
print(np.squeeze(\
    unifiedvocabulary_totaloccurrencecounts[\
    highest_totaloccurrences_indices[0:100]]))
```

```
['the' ',' '.' 'and' 'be' 'to' 'of' 'a' 'â\x80\x9d' 'in' 'have' 'i' 'he'
 'you' 'her' 'she' 'they' 'it' 'that' 'say' 'dorothy' 'as' 'so' 'for'
 'not' 'with' 'at' 'but' 'do' 'all' 'them' 'scarecrow' 'his' ';' '?' 'him'
 'my' 'me' 'woodman' 'come' 'on' 'lion' 'oz' 'great' 'make' 'will' 'go'
 'â\x80\x9ci' 'when' 'little' 'tin' 'witch' 'ask' 'this' 'one' 'could'
 'from' 'see' 'would' 'then' 'there' 'we' 'if' 'who' 'get' 'green' 'out'
 'up' 'can' 'their' 'head' 'look' 'know' 'no' 'think' 'girl' 'back' 'toto'
 '!' 'down' 'by' 'upon' 'answer' 'find' 'shall' 'again' 'city' 'give'
 'into' 'over' 'very' 'must' 'wicked' 'now' 'emerald' 'man' 'good' 'where'
 'after' 'walk']
[2709. 2494. 1485. 1458. 1324. 1009.  760.  737.  633.  441.  424.  422.
  410.  401.  369.  361.  357.  356.  342.  323.  321.  290.  282.  278.
  256.  252.  237.  228.  217.  217.  212.  203.  198.  185.  178.  172.
  172.  170.  168.  153.  151.  150.  148.  139.  139.  138.  137.  136.
  136.  133.  133.  128.  126.  118.  114.  113.  111.  110.  110.  106.
  105.  101.  100.   99.   98.   98.   95.   94.   92.   91.   90.   89.
   89.   88.   87.   85.   84.   83.   83.   82.   80.   78.   78.   76.
   76.   75.   73.   73.   73.   72.   72.   71.   71.   69.   67.   66.
   66.   65.   65.   65.]
```

[52]:
```
nltk.download('stopwords')

#%% Vocabulary pruning
nltkstopwords=nltk.corpus.stopwords.words('english')
pruningdecisions=np.zeros((len(unifiedvocabulary),1))
for k in range(len(unifiedvocabulary)):
    # Rule 1: check the nltk stop word list
    if (unifiedvocabulary[k] in nltkstopwords):
        pruningdecisions[k]=1
    # Rule 2: if the word is in the top 1% of frequent words
```

```
    #if (k in highest_totaloccurrences_indices[\
    #    0:int(np.floor(len(unifiedvocabulary)*0.01))]):
    #    pruningdecisions[k]=1
   # Rule 3: if the word occurs less than 4 times
    if(unifiedvocabulary_totaloccurrencecounts[k] < 4):
        pruningdecisions[k] = 1
    # Rule 4: if the word is too short
    if len(unifiedvocabulary[k])<2:
        pruningdecisions[k]=1
    # Rule 5: if the word is too long
    if len(unifiedvocabulary[k])>20:
        pruningdecisions[k]=1
    # Rule 6: if the word has unwanted characters
    # (here for simplicity only a-z allowed)
    if unifiedvocabulary[k].isalpha()==False:
        pruningdecisions[k]=1
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\danie\AppData\Roaming\nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[53]:
```
#%% Get indices of documents to remaining words
longestoldtopruned=[]
tempind=-1
for k in range(len(unifiedvocabulary)):
    if pruningdecisions[k]==0:
        tempind=tempind+1
        longestoldtopruned.append(tempind)
    else:
        longestoldtopruned.append(-1)
```

[54]:
```
#%% Create pruned texts
longestparagraph_prunedtexts=[]
longestmyindices_in_prunedvocabulary=[]
for k in range(len(longestpara)):
    print(k)
    temp_newindices=[]
    temp_newdoc=[]
    for l in range(len(longestpara[k])):
        temp_oldindex=myindices_in_unifiedvocabulary[k][l]
        temp_newindex=longestoldtopruned[temp_oldindex]
        if temp_newindex!=-1:
            temp_newindices.append(temp_newindex)
            temp_newdoc.append(unifiedvocabulary[temp_oldindex])
    longestparagraph_prunedtexts.append(temp_newdoc)
    longestmyindices_in_prunedvocabulary.append(temp_newindices)
```

```
0
```

```
␣
↪---------------------------------------------------------------------------

    IndexError                                Traceback (most recent call␣
↪last)

    <ipython-input-54-c8cd775aa87a> in <module>()
      7     temp_newdoc=[]
      8     for l in range(len(longestpara[k])):
----> 9         temp_oldindex=myindices_in_unifiedvocabulary[k][l]
     10         temp_newindex=longestoldtopruned[temp_oldindex]
     11         if temp_newindex!=-1:


    IndexError: index 0 is out of bounds for axis 0 with size 0
```

[55]:
```
print('Top 100 word-list after pruning unified vocabulary:\n')
remaining_indices = np.squeeze(np.where(pruningdecisions==0)[0])
remaining_vocabulary = unifiedvocabulary[remaining_indices]
remainingvocabulary_totaloccurrencecounts =␣
 ↪unifiedvocabulary_totaloccurrencecounts[remaining_indices]
remaining_highest_totaloccurrences_indices = np.
 ↪argsort(-1*remainingvocabulary_totaloccurrencecounts, axis=0)
print(np.
 ↪squeeze(remaining_vocabulary[remaining_highest_totaloccurrences_indices[0:
 ↪100]]))
print(np.
 ↪squeeze(remainingvocabulary_totaloccurrencecounts[remaining_highest_totaloccurrences_indice
 ↪100]]
```

```
      File "<ipython-input-55-aed8dc634baa>", line 7
    print(np.
↪squeeze(remainingvocabulary_totaloccurrencecounts[remaining_highest_totaloccurrences_indices
↪100]]
                                                                                ␣
↪                                    ^
  SyntaxError: unexpected EOF while parsing
```

[56]:
```
n_docs=len(longestparagraph_prunedtexts)
n_vocab=len(remaining_vocabulary)
```

```
# Matrix of term frequencies
tfmatrix=scipy.sparse.lil_matrix((n_docs,n_vocab))
# Row vector of document frequencies
dfvector=scipy.sparse.lil_matrix((1,n_vocab))
# Loop over documents
for k in range(n_docs):
    # 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(paragraph_prunedtexts[k])):
        # Add current word to term-frequency count and document-count
        currentword=longestmyindices_in_prunedvocabulary[k][l]
        tfmatrix[k,currentword]=tfmatrix[k,currentword]+1
        temp_dfvector[0,currentword]=1
    # Add which words occurred in this document to overall document counts
    dfvector=dfvector+temp_dfvector

# TF:length-normalized frequency
for i in range(n_docs):
    for j in range(len(tfmatrix.data[i])):
        tfmatrix.data[i][j] = tfmatrix.data[i][j]/len(tfmatrix.data[i])


# smoothed logarithmic idf
idfvector = np.squeeze(np.array(dfvector.todense()))
idfvector = np.log(1 + ((idfvector+1)**-1)*n_docs)

tfidfmatrix = scipy.sparse.lil_matrix((n_docs, n_vocab))
for k in range(n_docs):
    # tf and idf terms
    tfidfmatrix[k,:]=tfmatrix[k,:]*idfvector

# tf-idf matrix
#tfidfmatrix = scipy.sparse.lil_matrix((n_docs, n_vocab))
for k in range(n_docs):
    # find nonzero term frequencies
    tempindices = np.nonzero(tfmatrix[k, :])[1]
    tfterm = np.squeeze(np.array(tfmatrix[k, tempindices].todense()))
    tfidfmatrix[k, tempindices] = tfterm * idfvector[tempindices]
```

[57]:
```
#%% Use the TF-IDF matrix as data to be clustered
X=tfidfmatrix
# Normalize the documents to unit vector norm
tempnorms=np.squeeze(np.array(np.sum(X.multiply(X),axis=1)))
# If any documents have zero norm, avoid dividing them by zero
tempnorms[tempnorms==0]=1
X=scipy.sparse.diags(tempnorms**-0.5).dot(X)
```

```
n_data=np.shape(X)[0]
n_dimensions=np.shape(X)[1]
```

[58]:
```
# Function to initialize the Gaussian mixture model, create component parameters
def initialize_mixturemodel(X,n_components):
    # Create lists of sparse matrices to hold the parameters
    n_dimensions=np.shape(X)[1]
    n_data = np.shape(X)[0]
    mixturemodel_means=scipy.sparse.lil_matrix((n_components,n_dimensions))
    mixturemodel_weights=np.zeros((n_components))
    mixturemodel_covariances=[]
    mixturemodel_inversecovariances=[]
    for k in range(n_components):
        tempcovariance=scipy.sparse.lil_matrix((n_dimensions,n_dimensions))
        mixturemodel_covariances.append(tempcovariance)
        tempinvcovariance=scipy.sparse.lil_matrix((n_dimensions,n_dimensions))
        mixturemodel_inversecovariances.append(tempinvcovariance)
    # Initialize the parameters
    for k in range(n_components):
        mixturemodel_weights[k]=1/n_components
        # Pick a random data point as the initial mean
        tempindex=scipy.stats.randint.rvs(low=0,high=n_data)
        mixturemodel_means[k]=X[tempindex,:].toarray()
        # Initialize the covariance matrix to be spherical
        for l in range(n_dimensions):
            mixturemodel_covariances[k][l,l]=1
            mixturemodel_inversecovariances[k][l,l]=1
    ␣
 ↪return(mixturemodel_weights,mixturemodel_means,mixturemodel_covariances,mixturemodel_invers
```

[59]:
```
def␣
 ↪run_estep(X,mixturemodel_means,mixturemodel_covariances,mixturemodel_inversecovariances,mix
 ↪
    # For each component, compute terms that do not involve data
    meanterms=np.zeros((n_components))
    logdeterminants=np.zeros((n_components))
    logconstantterms=np.zeros((n_components))

    for k in range(n_components):
        # Compute mu_k*inv(Sigma_k)*mu_k
        meanterms[k]=(mixturemodel_means[k,:
 ↪]*mixturemodel_inversecovariances[k]*mixturemodel_means[k,:].T)[0,0]
        # Compute determinant of Sigma_k. For a diagonal matrix
        # this is just the product of the main diagonal
        logdeterminants[k]=np.sum(np.log(mixturemodel_covariances[k].
 ↪diagonal(0)))
        # Compute constant term beta_k * 1/(|Sigma_k|^1/2)
```

```
        # Omit the (2pi)^d/2 as it cancels out
        logconstantterms[k]=np.log(mixturemodel_weights[k]) - 0.
↪5*logdeterminants[k]

    print('E-step part2 ')
    # Compute terms that involve distances of data from components
    xnorms=np.zeros((n_data,n_components))
    xtimesmu=np.zeros((n_data,n_components))

    for k in range(n_components):
        #print(k)
        xnorms[:,k]=(X*mixturemodel_inversecovariances[k]*X.T).diagonal(0)
        xtimesmu[:,k]=np.squeeze((X*mixturemodel_inversecovariances[k]*␣
↪mixturemodel_means[k,:].T).toarray())

    xdists=xnorms+np.matlib.repmat(meanterms,n_data,1)-2*xtimesmu
    # Substract maximal term before exponent (cancels out) to maintain␣
↪computational precision
    numeratorterms=logconstantterms-xdists/2
    numeratorterms-=np.matlib.repmat(np.
↪max(numeratorterms,axis=1),n_components,1).T
    numeratorterms=np.exp(numeratorterms)
    mixturemodel_componentmemberships=numeratorterms/np.matlib.repmat(np.
↪sum(numeratorterms,axis=1),n_components,1).T
    return(mixturemodel_componentmemberships)
```

[60]:
```
def run_mstep_sumweights(mixturemodel_componentmemberships):
    # Compute total weight per component
    mixturemodel_weights=np.sum(mixturemodel_componentmemberships,axis=0)
    return(mixturemodel_weights)
```

[61]:
```
def run_mstep_means(X,mixturemodel_componentmemberships,mixturemodel_weights):
    # Update component means
    mixturemodel_means=scipy.sparse.lil_matrix((n_components,n_dimensions))
    for k in range(n_components):
        mixturemodel_means[k,:]=\
            np.sum(scipy.sparse.diags(mixturemodel_componentmemberships[:,k]).
↪dot(X),axis=0)
        mixturemodel_means[k,:]/=mixturemodel_weights[k]
    return(mixturemodel_means)
```

[62]:
```
def␣
↪run_mstep_covariances(X,mixturemodel_componentmemberships,mixturemodel_weights,mixturemodel
↪
    # Update diagonal component covariance matrices
    n_dimensions=np.shape(X)[1]
```

```
    n_components=np.shape(mixturemodel_componentmemberships)[1]
    tempcovariances=np.zeros((n_components,n_dimensions))
    mixturemodel_covariances=[]
    mixturemodel_inversecovariances=[]

    for k in range(n_components):
        tempcovariances[k,:]= np.sum(scipy.sparse.diags(
            mixturemodel_componentmemberships[:,k]).dot(
            X.multiply(X)),axis=0)-mixturemodel_means[k,:].
↪multiply(mixturemodel_means[k,:])*mixturemodel_weights[k]
        tempcovariances[k,:]/=mixturemodel_weights[k]
        # Convert to sparse matrices
        tempepsilon=1e-10
        # Add a small regularization term
        temp_covariance=scipy.sparse.diags(tempcovariances[k,:]+tempepsilon)
        temp_inversecovariance=scipy.sparse.diags((tempcovariances[k,:
↪]+tempepsilon)**-1)
        mixturemodel_covariances.append(temp_covariance)
        mixturemodel_inversecovariances.append(temp_inversecovariance)
    return(mixturemodel_covariances,mixturemodel_inversecovariances)
```

[63]:
```
def run_mstep_normalizeweights(mixturemodel_weights):
    # Update mixture-component prior probabilities
    mixturemodel_weights/=sum(mixturemodel_weights)
    return(mixturemodel_weights)
```

[64]:
```
#%% Perform the EM algorithm iterations
def perform_emalgorithm(X,n_components,n_emiterations):
    mixturemodel_weights,mixturemodel_means,mixturemodel_covariances,\
    mixturemodel_inversecovariances=initialize_mixturemodel(X,n_components)
    for t in range(n_emiterations):
        # ====== E-step: Compute the component membership
        # probabilities of each data point ======
        print('E-step ' + str(t))
        ␣
↪mixturemodel_componentmemberships=run_estep(X,mixturemodel_means,mixturemodel_covariances,\
        mixturemodel_inversecovariances,mixturemodel_weights)
        # ====== M-step: update component parameters======
        print('M-step ' + str(t))
        print('M-step part1 ' + str(t))
        ␣
↪mixturemodel_weights=run_mstep_sumweights(mixturemodel_componentmemberships)
        print('M-step part2 ' + str(t))
        ␣
↪mixturemodel_means=run_mstep_means(X,mixturemodel_componentmemberships,mixturemodel_weights
        print('M-step part3 ' + str(t))
```

```
       ␣
↪mixturemodel_covariances,mixturemodel_inversecovariances=run_mstep_covariances(X,\
       ␣
↪mixturemodel_componentmemberships,mixturemodel_weights,mixturemodel_means)
        print('M-step part4 ' + str(t))
        mixturemodel_weights=run_mstep_normalizeweights(mixturemodel_weights)
    return(mixturemodel_weights,mixturemodel_means,mixturemodel_covariances,\
mixturemodel_inversecovariances)
# Try out the functions we just defined on the data
n_components=10
n_emiterations=100
mixturemodel_weights,mixturemodel_means,mixturemodel_covariances,\
mixturemodel_inversecovariances =␣
↪perform_emalgorithm(X,n_components,n_emiterations)
```

```
       ␣
↪---------------------------------------------------------------------------

       ValueError                                Traceback (most recent call␣
↪last)

       <ipython-input-64-b8e189326e21> in <module>()
        21 n_components=10
        22 n_emiterations=100
   ---> 23␣
↪mixturemodel_weights,mixturemodel_means,mixturemodel_covariances,mixturemodel_inversecovaria
↪= perform_emalgorithm(X,n_components,n_emiterations)


       <ipython-input-64-b8e189326e21> in perform_emalgorithm(X, n_components,␣
↪n_emiterations)
         1 #%% Perform the EM algorithm iterations
         2 def perform_emalgorithm(X,n_components,n_emiterations):
   ----> 3     ␣
↪mixturemodel_weights,mixturemodel_means,mixturemodel_covariances,    ␣
↪mixturemodel_inversecovariances=initialize_mixturemodel(X,n_components)
         4     for t in range(n_emiterations):
         5         # ====== E-step: Compute the component membership


       <ipython-input-58-8993e11d61c8> in initialize_mixturemodel(X,␣
↪n_components)
        17         mixturemodel_weights[k]=1/n_components
        18         # Pick a random data point as the initial mean
   ---> 19         tempindex=scipy.stats.randint.rvs(low=0,high=n_data)
        20         mixturemodel_means[k]=X[tempindex,:].toarray()
```

```
     21         # Initialize the covariance matrix to be spherical


    ~\Anaconda3\lib\site-packages\scipy\stats\_distn_infrastructure.py in␣
↪rvs(self, *args, **kwargs)
   2807         """
   2808         kwargs['discrete'] = True
-> 2809         return super(rv_discrete, self).rvs(*args, **kwargs)
   2810
   2811     def pmf(self, k, *args, **kwds):


    ~\Anaconda3\lib\site-packages\scipy\stats\_distn_infrastructure.py in␣
↪rvs(self, *args, **kwds)
    938         cond = logical_and(self._argcheck(*args), (scale >= 0))
    939         if not np.all(cond):
--> 940             raise ValueError("Domain error in arguments.")
    941
    942         if np.all(scale == 0):


    ValueError: Domain error in arguments.
```

[65]:
```
for k in range(n_components):
    print(k)
    highest_dimensionweight_indices=np.argsort(-np.
↪squeeze(mixturemodel_means[k,:].toarray()),axis=0)

    print(' '.join(remaining_vocabulary[highest_dimensionweight_indices[1:10]]))
```

```
0
stranger gravely grateful gown street stretch gold strike glass
1
party pass pat pave perch person pick part piece
2
nearly nearer near mouth monster milkmaid mile midst middle
3
place play pleasant please pleased plenty point pole poor
4
often oil open order others ought palace offer part
5
ought paint party pat patch pave perch pick piece
6
mistake tire tip monster morning mouth move tiny munchkin
7
rock gently gaze gayelette gather roof funny rope fruit
```

```
8
inquire ruler remark dear spider figure five scare inside
9
inquire indeed immediately imagine spider hung humbug spite spot
```

[66]:

```
# Version 2 - Get documents closest to component mean, i.e. highest p(d|k).
# ---The computation of distances here is the same as done in the E-step of␣
↪EM---
# For each component, compute terms that do not involve data
meanterms=np.zeros((n_components))
logdeterminants=np.zeros((n_components))
logconstantterms=np.zeros((n_components))
for k in range(n_components):
    # Compute mu_k*inv(Sigma_k)*mu_k
    meanterms[k]=(mixturemodel_means[k,:
↪]*mixturemodel_inversecovariances[k]*mixturemodel_means[k,:].T)[0,0]

# Compute terms that involve distances of data from components
xnorms=np.zeros((n_data,n_components))
xtimesmu=np.zeros((n_data,n_components))

for k in range(n_components):
    xnorms[:,k]=(X*mixturemodel_inversecovariances[k]*X.T).diagonal(0)
    xtimesmu[:,k]=np.
↪squeeze((X*mixturemodel_inversecovariances[k]*mixturemodel_means[k,:].T).
↪toarray())

xdists=xnorms+np.matlib.repmat(meanterms,n_data,1)-2*xtimesmu

for k in range(n_components):
    tempdists=np.array(np.squeeze(xdists[:,k]))
    highest_componentprob_indices=np.argsort(-1*tempdists,axis=0)
    print(k)
    print(highest_componentprob_indices[0:10])
    print(' '.join(paragraph_nltk_texts[highest_componentprob_indices[0]]))
```

```
     ␣
↪---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call␣
↪last)

        <ipython-input-66-a71fcfb83997> in <module>()
         14
         15 for k in range(n_components):
```

```
   ---> 16     xnorms[:,k]=(X*mixturemodel_inversecovariances[k]*X.T).
↪diagonal(0)
        17     xtimesmu[:,k]=np.
↪squeeze((X*mixturemodel_inversecovariances[k]*mixturemodel_means[k,:].T).
↪toarray())
        18


       ~\Anaconda3\lib\site-packages\scipy\sparse\compressed.py in␣
↪diagonal(self, k)
       517         rows, cols = self.shape
       518         if k <= -rows or k >= cols:
   --> 519             raise ValueError("k exceeds matrix dimensions")
       520         fn = getattr(_sparsetools, self.format + "_diagonal")
       521         y = np.empty(min(rows + min(k, 0), cols - max(k, 0)),


       ValueError: k exceeds matrix dimensions
```

```
[ ]:
```