Ahmad Sharif

K436765

DATA.STAT.840 Statistical Methods for Text Data Analysis

In [30]:
```
import nltk
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
from scipy.sparse import csr_matrix
from sklearn.decomposition import TruncatedSVD
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer

import scipy.stats
nltk.download('stopwords')

# Load stopwords
stop_words = set(stopwords.words('english'))

# (a) Download the 20 Newsgroups data set from http://qwone.com/~jason/20Newsgroups/
from sklearn.datasets import fetch_20newsgroups
```

```
[nltk_data] Downloading package stopwords to /home/ahmad/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [10]:
```
''' b) In this exercise we consider only four of the newsgroups: rec.autos, rec.moto
rec.sport.baseball, and rec.sport.hockey. Process the documents of the four newsgrou
using the pipeline described on the lectures, including vocabulary pruning. '''

categories = ['rec.autos', 'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.hocke
dataset = fetch_20newsgroups(subset='all', shuffle=True, random_state=42, categories
corpus = dataset.data


dataset

''' c) Create a TF-IDF representation for the documents, using Length-normalized fre
and Smoothed logarithmic inverse document frequency (IDF). '''

# TF-IDF Vectorization
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
tfidf_matrix = vectorizer.fit_transform(corpus)

# Vocabulary pruning using LSA
n_low_dimensions = 10
lsa_model = TruncatedSVD(n_components=n_low_dimensions)
X_small = lsa_model.fit_transform(tfidf_matrix)


print(lsa_model.singular_values_)

# Examine a factor (here the one with the largest singular value)
print(lsa_model.components_[0, :])

# 20 words with the largest absolute weights in the factor
top_weights_indices = np.argsort(-1 * np.abs(lsa_model.components_[0, :]))
feature_names = np.array(vectorizer.get_feature_names_out())
print(feature_names[top_weights_indices[0:20]])
```

```
[10.31839701  5.69499476  4.75723351  4.44560893  4.33675234  4.169413
  4.09657462  3.96486213  3.85453629  3.83934871]
[0.01759519 0.02251091 0.00257834 ... 0.01100637 0.00228007 0.01042266]
['edu' 'com' 'writes' 'ca' 'article' 'game' 'subject' 'don' 'organization'
 'lines' 'car' 'like' 'university' 'just' 'team' 'posting' 'nntp' 'host'
 'year' 'think']
```

In [3]:

```python
'''  c) Create a TF-IDF representation for the documents, using Length-normalized fre
and Smoothed logarithmic inverse document frequency (IDF). '''


def plsa(document_to_word_matrix, n_topics, n_iterations):
    n_docs, n_vocab = np.shape(document_to_word_matrix)


    theta = np.random.uniform(size=(n_vocab, n_topics))
    theta /= np.tile(np.sum(theta, axis=0), (n_vocab, 1))

    psi = np.random.uniform(size=(n_topics, n_docs))
    psi /= np.tile(np.sum(psi, axis=0), (n_topics, 1))

    n_words_in_docs = np.squeeze(np.array(np.sum(document_to_word_matrix, axis=1)))
    n_total_words = np.sum(n_words_in_docs)
    pi = n_words_in_docs / n_total_words

    for _ in range(n_iterations):
        # E-step
        doc_word_to_topics = []
        doc_word_to_topic_sum = np.zeros((n_docs, n_vocab))
        for t in range(n_topics):
            doc_word_to_topict = np.tile(theta[:, t], (n_docs, 1)) * np.tile(psi[t,
            epsilon = 1e-14
            doc_word_to_topict += epsilon
            doc_word_to_topics.append(doc_word_to_topict)
            doc_word_to_topic_sum += doc_word_to_topict

        for t in range(n_topics):
            doc_word_to_topics[t] /= doc_word_to_topic_sum

        # M-step
        for t in range(n_topics):
            psi[t, :] = np.squeeze(np.array(np.sum(
                np.multiply(document_to_word_matrix + epsilon, doc_word_to_topics[t]
        psi /= np.tile(np.sum(psi, axis=0), (n_topics, 1))

        for t in range(n_topics):
            theta[:, t] = np.squeeze(np.array(np.sum(
                np.multiply(document_to_word_matrix, doc_word_to_topics[t]), axis=0)
        theta /= np.tile(np.sum(theta, axis=0), (n_vocab, 1))

    return pi, psi, theta


# TF-IDF Vectorization with Length-normalized frequency (TF) and Smoothed logarithmi
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
tfidf_matrix = vectorizer.fit_transform(corpus)
tfidf_matrix = csr_matrix(tfidf_matrix)  # Convert to sparse matrix

# PLSA
n_topics = 10
n_iterations = 50
pi, psi, theta = plsa(tfidf_matrix, n_topics, n_iterations)
```

```
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call last)
Cell In[3], line 55
     53 n_topics = 10
     54 n_iterations = 50
---> 55 pi, psi, theta = plsa(tfidf_matrix, n_topics, n_iterations)

Cell In[3], line 36, in plsa(document_to_word_matrix, n_topics, n_iterations)
     33 # M-step
     34 for t in range(n_topics):
     35     psi[t, :] = np.squeeze(np.array(np.sum(
---> 36         np.multiply(document_to_word_matrix + epsilon, doc_word_to_topics
[t]), axis=1)))
     37 psi /= np.tile(np.sum(psi, axis=0), (n_topics, 1))
     39 for t in range(n_topics):

File ~/anaconda3/lib/python3.11/site-packages/scipy/sparse/_base.py:467, in spmatrix.
__add__(self, other)
    465         return self.copy()
    466     # Now we would add this scalar to every element.
--> 467     raise NotImplementedError('adding a nonzero scalar to a '
    468                               'sparse matrix is not supported')
    469 elif isspmatrix(other):
    470     if other.shape != self.shape:

NotImplementedError: adding a nonzero scalar to a sparse matrix is not supported
```

In [11]:
```
''' d) Apply latent semantic analysis to the TF-IDF matrix, to find 10 underlying fa



n_topics = 10

# Apply TruncatedSVD for LSA
lsa_model = TruncatedSVD(n_components=n_topics)
lsa_matrix = lsa_model.fit_transform(tfidf_matrix)

# Examine the factors
print(lsa_model.singular_values_)  # Singular values
print(lsa_model.components_)        # Factors

# Access the transformed TF-IDF
print(lsa_matrix)
```

```
[10.31839701  5.69499716  4.75716961  4.44274412  4.337713    4.17673975
  4.09626038  3.96767267  3.85081661  3.82714268]
[[ 0.01759533  0.02251085  0.00257856 ...  0.01100638  0.00228009
   0.01042255]
 [ 0.00541773 -0.01817195 -0.00504579 ...  0.00788033  0.00504621
  -0.02207526]
 [ 0.00695189 -0.0215263   0.00444565 ...  0.00614491  0.00564106
   0.01453824]
 ...
 [-0.02312671 -0.00900819 -0.00438567 ...  0.00242274 -0.00435804
   0.02864854]
 [ 0.01498695  0.02171728  0.00140785 ... -0.00126983  0.00900423
  -0.03144969]
 [-0.01192349  0.01080683 -0.01545888 ... -0.00283797  0.00045932
   0.05358575]]
[[ 0.14863628  0.10713103  0.11191805 ... -0.04761222  0.05171661
   0.00891934]
 [ 0.17835921 -0.03078103 -0.07558104 ... -0.07305895 -0.11017075
  -0.11658226]
 [ 0.22019046  0.12428973  0.01533644 ...  0.07663957 -0.09047817
  -0.01684534]
 ...
 [ 0.12762034 -0.01895873  0.02862286 ... -0.02250516  0.0141198
  -0.00151372]
 [ 0.11245082  0.04849857  0.04339154 ... -0.0069751  -0.0078708
   0.0091013 ]
 [ 0.0998395  -0.05067188 -0.00074359 ...  0.07076911  0.09786765
  -0.02503128]]
```

In [15]:
```
''' e) (e) Describe the resulting factors: list the 10 words with highest (absolute)
factor. Do the factors seem related to individual newsgroups? Does their content seem
meaningful? '''


# Get the feature names from the TF-IDF vectorizer
feature_names = np.array(vectorizer.get_feature_names_out())

# Get the words with the highest absolute weight in each factor
for i in range(n_topics):
    factor_weights = lsa_model.components_[i]
    top_word_indices = np.argsort(np.abs(factor_weights))[::-1][:10]
    top_words = feature_names[top_word_indices]

    print(f"Factor {i + 1}:")
    print(top_words)
    print()

# It seems meaningful. However, some nouns are dominant tokens like 'cunixb'.
```

```
Factor 1:
['edu' 'com' 'writes' 'ca' 'article' 'game' 'subject' 'don' 'organization'
 'lines']

Factor 2:
['com' 'game' 'car' 'team' 'bike' 'hockey' 'games' 'sun' 'espn' 'win']

Factor 3:
['edu' 'com' 'sun' 'ca' 'east' 'car' 'state' 'ohio' 'university' 'ed']

Factor 4:
['sun' 'car' 'edu' 'east' 'ed' 'green' 'columbia' 'gld' 'espn' 'egreen']

Factor 5:
['ca' 'com' 'baseball' 'car' 'edu' 'sun' 'bnr' 'jewish' 'braves' 'east']

Factor 6:
['ca' 'car' 'cs' 'maynard' 'roger' 'bike' 'laurentian' 'sun' 'espn' 'com']

Factor 7:
['sun' 'nec' 'behanna' 'east' 'car' 'green' 'nj' 'ed' 'com' 'maynard']

Factor 8:
['columbia' 'gld' 'espn' 'cunixb' 'cc' 'dare' 'gary' 'game' 'ohio'
 'andrew']

Factor 9:
['columbia' 'gld' 'cmu' 'game' 'cc' 'andrew' 'cunixb' 'hp' 'behanna' 'nec']

Factor 10:
['behanna' 'nec' 'ohio' 'state' 'magnus' 'cmu' 'andrew' 'acs' 'nj' 'syl']
```

In [16]:

```
''' (f) Do the same with 15 factors (the first 10 factors will be the same). Do the 
seem more or less meaningful? '''

# Get the feature names from the TF-IDF vectorizer
feature_names = np.array(vectorizer.get_feature_names_out())

# Get the words with the highest absolute weight in each factor
for i in range(n_topics):
    factor_weights = lsa_model.components_[i]
    top_word_indices = np.argsort(np.abs(factor_weights))[::-1][:15]
    top_words = feature_names[top_word_indices]

    print(f"Factor {i + 1}:")
    print(top_words)
    print()

# More meaningful
```

```
Factor 1:
['edu' 'com' 'writes' 'ca' 'article' 'game' 'subject' 'don' 'organization'
 'lines' 'car' 'like' 'university' 'just' 'team']

Factor 2:
['com' 'game' 'car' 'team' 'bike' 'hockey' 'games' 'sun' 'espn' 'win'
 'dod' 'players' 'play' 'season' 'year']

Factor 3:
['edu' 'com' 'sun' 'ca' 'east' 'car' 'state' 'ohio' 'university' 'ed'
 'andrew' 'cc' 'green' 'team' 'uiuc']

Factor 4:
['sun' 'car' 'edu' 'east' 'ed' 'green' 'columbia' 'gld' 'espn' 'egreen'
 'cc' 'cmu' 'andrew' 'year' 'buffalo']

Factor 5:
['ca' 'com' 'baseball' 'car' 'edu' 'sun' 'bnr' 'jewish' 'braves' 'east'
 'hockey' 'canada' 'runs' 'bike' 'netcom']

Factor 6:
['ca' 'car' 'cs' 'maynard' 'roger' 'bike' 'laurentian' 'sun' 'espn' 'com'
 'edu' 'game' 'ramsey' 'duke' 'dod']

Factor 7:
['sun' 'nec' 'behanna' 'east' 'car' 'green' 'nj' 'ed' 'com' 'maynard'
 'syl' 'egreen' 'bike' 'ca' '11']

Factor 8:
['columbia' 'gld' 'espn' 'cunixb' 'cc' 'dare' 'gary' 'game' 'ohio'
 'andrew' 'magnus' 'state' 'cmu' 'behanna' 'nec']

Factor 9:
['columbia' 'gld' 'cmu' 'game' 'cc' 'andrew' 'cunixb' 'hp' 'behanna' 'nec'
 'dare' 'gary' 'espn' 'pens' 'games']

Factor 10:
['behanna' 'nec' 'ohio' 'state' 'magnus' 'cmu' 'andrew' 'acs' 'nj' 'syl'
 'sun' 'chris' 'bnr' 'uk' 'list']
```

In [20]:

```python
''' (a) Using the same data as in Exercise 6.1 (four newsgroups), create a term freq
raw term counts for the documents. '''

corpus = dataset.data

# Create the CountVectorizer
count_vectorizer = CountVectorizer(stop_words='english', max_features=5000)

# Fit and transform the documents to get the term frequency matrix
tf_matrix = count_vectorizer.fit_transform(corpus)

# Convert the sparse matrix to a dense matrix if needed
tf_matrix_dense = tf_matrix.todense()

# Display the term frequency matrix
print(tf_matrix_dense)
```

```
[[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]]
```

In [24]:

```
''' c) Describe the resulting factors: list the 10 words with highest probability in

top_words_indices = np.argsort(-1 * theta, axis=0)

# Print the top 10 words for each factor
for i in range(n_topics_plsa):
    print(f"Factor {i + 1}:")
    top_words = feature_names[top_words_indices[:, i]][:10]
    print(", ".join(top_words))
    print()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[24], line 3
      1 ''' c) Describe the resulting factors: list the 10 words with highest probabi
lity in each factor. '''
----> 3 top_words_indices = np.argsort(-1 * theta, axis=0)
      5 # Print the top 10 words for each factor
      6 for i in range(n_topics_plsa):

NameError: name 'theta' is not defined
```

In [27]:

```
''' 6.3.1 (a)create a term frequency matrix of raw term counts for the documents. ''

corpus = dataset.data  # This line is already provided in your code

# Create a CountVectorizer
count_vectorizer = CountVectorizer(stop_words='english', max_features=5000)  # You ca

# Fit and transform the documents to obtain the term frequency matrix
tf_matrix = count_vectorizer.fit_transform(corpus)

# Get the feature names (words) corresponding to the columns of the matrix
feature_names = count_vectorizer.get_feature_names_out()

# Convert the sparse matrix to a dense matrix if needed
tf_matrix_dense = tf_matrix.toarray()
print('tf_matrix : ', tf_matrix)
print('tf_matrix_dense : ', tf_matrix_dense)
```

```
tf_matrix :    (0, 1995)        3
  (0, 2276)     3
  (0, 4721)     5
  (0, 905)      5
  (0, 2090)     2
  (0, 670)      2
  (0, 4340)     1
  (0, 3461)     1
  (0, 2969)     1
  (0, 4260)     3
  (0, 3163)     1
  (0, 3496)     1
  (0, 2255)     1
  (0, 3749)     1
  (0, 3275)     1
  (0, 4681)     1
  (0, 4766)     2
  (0, 706)      1
  (0, 938)      1
  (0, 2717)     1
  (0, 180)      1
  (0, 585)      2
  (0, 3168)     2
  (0, 1830)     2
  (0, 4946)     2
  :      :
  (3978, 2947)  1
  (3978, 1005)  1
  (3978, 3683)  1
  (3978, 2189)  1
  (3978, 1055)  1
  (3978, 2465)  1
  (3978, 3318)  1
  (3978, 3818)  1
  (3978, 2136)  1
  (3978, 87)    1
  (3978, 346)   1
  (3978, 3341)  1
  (3978, 1156)  1
  (3978, 3837)  1
  (3978, 784)   1
  (3978, 1410)  1
  (3978, 2475)  1
  (3978, 2164)  1
  (3978, 4919)  1
  (3978, 3920)  1
  (3978, 886)   1
  (3978, 3663)  1
  (3978, 1332)  1
  (3978, 986)   1
  (3978, 4986)  3
tf_matrix_dense :  [[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]]
```

In [32]:
```
''' b) Apply Latent Dirichlet Allocation to the term frequency matrix to find 10 und

n_topics = 10
n_iterations = 200
```

```
pi_plsa, psi_plsa, theta_plsa = plsa(tf_matrix_dense, n_topics, n_iterations)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[32], line 7
      4 n_iterations = 200
      6 # Apply PLSA to the term frequency matrix
----> 7 pi_plsa, psi_plsa, theta_plsa = plsa(tf_matrix_dense, n_topics, n_iterations)

Cell In[21], line 10, in plsa(document_to_word_matrix, n_topics, n_iterations)
      8 # Initialize theta and psi with random values
      9 theta = np.random.rand(n_vocab, n_topics)
---> 10 theta /= np.matlib.repmat(np.sum(theta, axis=0), n_vocab, 1)
     12 psi = np.random.rand(n_topics, n_docs)
     13 psi /= np.matlib.repmat(np.sum(psi, axis=0), n_topics, 1)

File ~/anaconda3/lib/python3.11/site-packages/numpy/__init__.py:320, in __getattr__(a
ttr)
    317     from .testing import Tester
    318     return Tester
--> 320 raise AttributeError("module {!r} has no attribute "
    321                      "{!r}".format(__name__, attr))

AttributeError: module 'numpy' has no attribute 'matlib'
```

In [33]:
```
''' c) Describe the resulting factors: list the 10 words with highest probability in
# Assuming 'feature_names' is the array of feature names obtained from the CountVect
# and 'theta_plsa' is the matrix obtained from PLSA

# Get the indices of the top words for each factor
top_words_indices = np.argsort(-1 * theta_plsa, axis=0)

# Print the top 10 words for each topic
for i in range(n_topics):
    print(f"Topic {i + 1}:")
    top_words = feature_names[top_words_indices[:, i]][:10]
    print(", ".join(top_words))
    print()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[33], line 6
      1 ''' c) '''
      2 # Assuming 'feature_names' is the array of feature names obtained from the Co
untVectorizer
      3 # and 'theta_plsa' is the matrix obtained from PLSA
      4 
      5 # Get the indices of the top words for each factor
----> 6 top_words_indices = np.argsort(-1 * theta_plsa, axis=0)
      8 # Print the top 10 words for each topic
      9 for i in range(n_topics):

NameError: name 'theta_plsa' is not defined
```

In [34]:
```
''' d) Find, for each topic, the document (message) with highest probability of that
its 100 first words. '''

# Assuming 'psi_plsa' is the matrix obtained from PLSA
# and 'corpus' is the list of documents

# Get the index of the document with the highest probability for each topic
top_doc_indices = np.argmax(psi_plsa, axis=1)

# Print the first 100 words of the top document for each topic
```

```python
for i, doc_index in enumerate(top_doc_indices):
    print(f"Topic {i + 1} - Document with Highest Probability:")
    top_doc = corpus[doc_index]
    # Print the first 100 words
    print(" ".join(top_doc.split()[:100]))
    print()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[34], line 7
      1 ''' d) '''
      3 # Assuming 'psi_plsa' is the matrix obtained from PLSA
      4 # and 'corpus' is the list of documents
      5 
      6 # Get the index of the document with the highest probability for each topic
----> 7 top_doc_indices = np.argmax(psi_plsa, axis=1)
      9 # Print the first 100 words of the top document for each topic
     10 for i, doc_index in enumerate(top_doc_indices):

NameError: name 'psi_plsa' is not defined
```