# CS145 Howework 6, Naive Bayes and Topic Modeling

**Due date:** HW6 is due on **11:59 PM PT, Dec. 14 (Monday, Final Week)**. Please submit through GradeScope.

---

## Print Out Your Name and UID

**Name: Danning Yu, UID: 305087992**

---

## Important Notes about HW6

- HW6, as the last homework, is optional if you choose to use the first 5 homework assignments for homework grading. We will select your highest 5 homework grades to calculate your final homework grade.
- Since HW6 is optional, for the implementaion of Naive Bayes and pLSA, you can choose to implement the provided `.py` and `.py` file by filling in the blocks. Alternatively, you are given the option to implement completely from scratch based on your understanding. Note that some packages with ready-to-use implementation of Naive Bayes and pLSA are not allowed.

---

## Before You Start

You need to first create HW6 conda environment by the given `cs145hw6.yml` file, which provides the name and necessary packages for this tasks. If you have `conda` properly installed, you may create, activate or deactivate by the following commands:

```
conda env create -f cs145hw6.yml
conda activate hw6
conda deactivate
```

OR

```
conda env create --name NAMEOFYOURCHOICE -f cs145hw6.yml
conda activate NAMEOFYOURCHOICE
conda deactivate
```

To view the list of your environments, use the following command:

```
conda env list
```

More useful information about managing environments can be found here.

You may also quickly review the usage of basic Python and Numpy package, if needed in coding for matrix operations.

In this notebook, you must not delete any code cells in this notebook. If you change any code outside the blocks (such as hyperparameters) that you are allowed to edit (between `START/END YOUR CODE HERE`), you need to highlight these changes. You may add some additional cells to help explain your results and observations.

In [1]:

```
import numpy as np
from numpy import zeros, int8, log
from pylab import random
import pandas as pd
import matplotlib.pyplot as plt
from pylab import rcParams
rcParams['figure.figsize'] = 8,8
import seaborn as sns; sns.set()
import re
import time
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize
from sklearn.metrics import confusion_matrix
%load_ext autoreload
%autoreload 2
```

```
[nltk_data] Downloading package punkt to /Users/danningy/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Note that `seaborn` in HW6 is only used for ploting classification confusion matrix (in a "heatmap" style). If you encounter installation problem and cannot solve it, you may use alternative plot methods to show your results.

## Section 1: Naive Bayes for Text (50 points)

Naive Bayers is one generative model for text classification. In the problem, you are given a document in `dataset` folder. The original data comes from "20 newsgroups". You can use the provided data files to save efforts on preprocessing.

Note: The code and dataset are under the subfolder named `nb`.

In [2]:

```
### Data processing and preparation
# read train/test labels from files
train_label = pd.read_csv('./nb/dataset/train.label',names=['t'])
train_label = train_label['t'].tolist()
test_label = pd.read_csv('./nb/dataset/test.label', names=['t'])
test_label= test_label['t'].tolist()

# read train/test documents from files
train_data = open('./nb/dataset/train.data')
df_train = pd.read_csv(train_data, delimiter=' ', names=['docIdx', 'wordIdx', 'c
test_data = open('./nb/dataset/test.data')
df_test = pd.read_csv(test_data, delimiter=' ', names=['docIdx', 'wordIdx', 'cou
```

```
# read vocab
vocab = open('./nb/dataset/vocabulary.txt')
vocab_df = pd.read_csv(vocab, names = ['word'])
vocab_df = vocab_df.reset_index()
vocab_df['index'] = vocab_df['index'].apply(lambda x: x+1)

# add label column to original df_train
docIdx = df_train['docIdx'].values
i = 0
new_label = []
for index in range(len(docIdx)-1):
    new_label.append(train_label[i])
    if docIdx[index] != docIdx[index+1]:
        i += 1
new_label.append(train_label[i])
df_train['classIdx'] = new_label
```

If you have the data prepared properly, the following line of code would return the head of the `df_train` dataframe, which is,

| | docIdx | wordIdx | count | classIdx |
|---|---|---|---|---|
| 0 | 1 | 1 | 4 | 1 |
| 1 | 1 | 2 | 2 | 1 |
| 2 | 1 | 3 | 10 | 1 |
| 3 | 1 | 4 | 4 | 1 |
| 4 | 1 | 5 | 2 | 1 |

In [3]:

```
# check the head of 'df_train'
print(df_train.head())
```

```
   docIdx  wordIdx  count  classIdx
0       1        1      4         1
1       1        2      2         1
2       1        3     10         1
3       1        4      4         1
4       1        5      2         1
```

Complete the implementation of Naive Bayes model for text classification `nbm.py` . After that, run `nbm_sklearn.py` , which uses `sklearn` to implement naive bayes model for text classification. (Note that the dataset is slightly different loaded in `nbm_sklearn.py` and also you don't need to change anything in `nbm_sklearn.py` and directly run it.)

If the implementation is correct, you can expect the results are generally close on both train set accuracy and test set accuracy.

In [4]:

```
from nb.nbm import NB_model

# model training
nbm = NB_model()
nbm.fit(df_train, train_label, vocab_df)
```

```
Prior Probability of each class:
```

```
1: 0.04259472890229834
2: 0.05155736977549028
3: 0.05075871860857219
4: 0.05208980388676901
5: 0.051024935664211554
6: 0.052533498979501284
7: 0.051646108794036735
8: 0.052533498979501284
9: 0.052888455053687104
10: 0.0527109770165942
11: 0.05306593309078002
12: 0.0527109770165942
13: 0.05244475996095483
14: 0.0527109770165942
15: 0.052622237998047744
16: 0.05315467210932647
17: 0.04836276510781791
18: 0.05004880646020055
19: 0.04117490460555506
20: 0.033365870973467035
Training class 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Training completed!
```

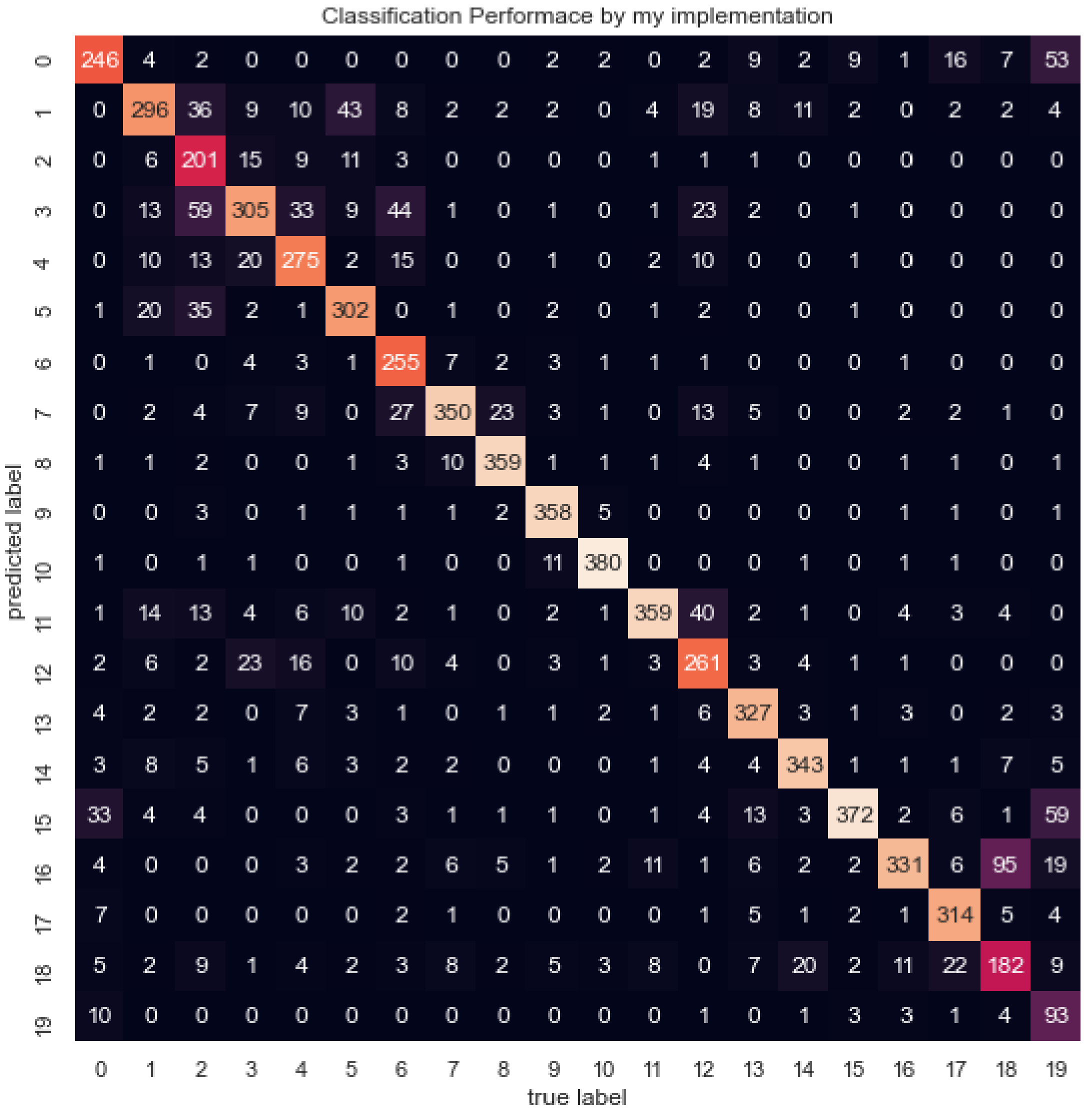
In [5]:

```
# make predictions on train set to validate the model
predict_train_labels = nbm.predict(df_train)
train_acc = (np.array(train_label) == np.array(predict_train_labels)).mean()
print("Accuracy on training data by my implementation: {}".format(train_acc))

# make predictions on test data
predict_test_labels = nbm.predict(df_test)
test_acc = (np.array(test_label) == np.array(predict_test_labels)).mean()
print("Accuracy on training data by my implementation: {}".format(test_acc))

test_res = np.array(test_label) == np.array(predict_test_labels)
for i, res in enumerate(test_res):
    if not res:
        print(f"Document {i+1} was misclassified")
        break
```

```
Finished predictions for doc 2500
Finished predictions for doc 5000
Finished predictions for doc 7500
Finished predictions for doc 10000
Accuracy on training data by my implementation: 0.9481764131688704
Finished predictions for doc 2500
Finished predictions for doc 5000
Finished predictions for doc 7500
Accuracy on training data by my implementation: 0.7873417721518987
Document 4 was misclassified
```

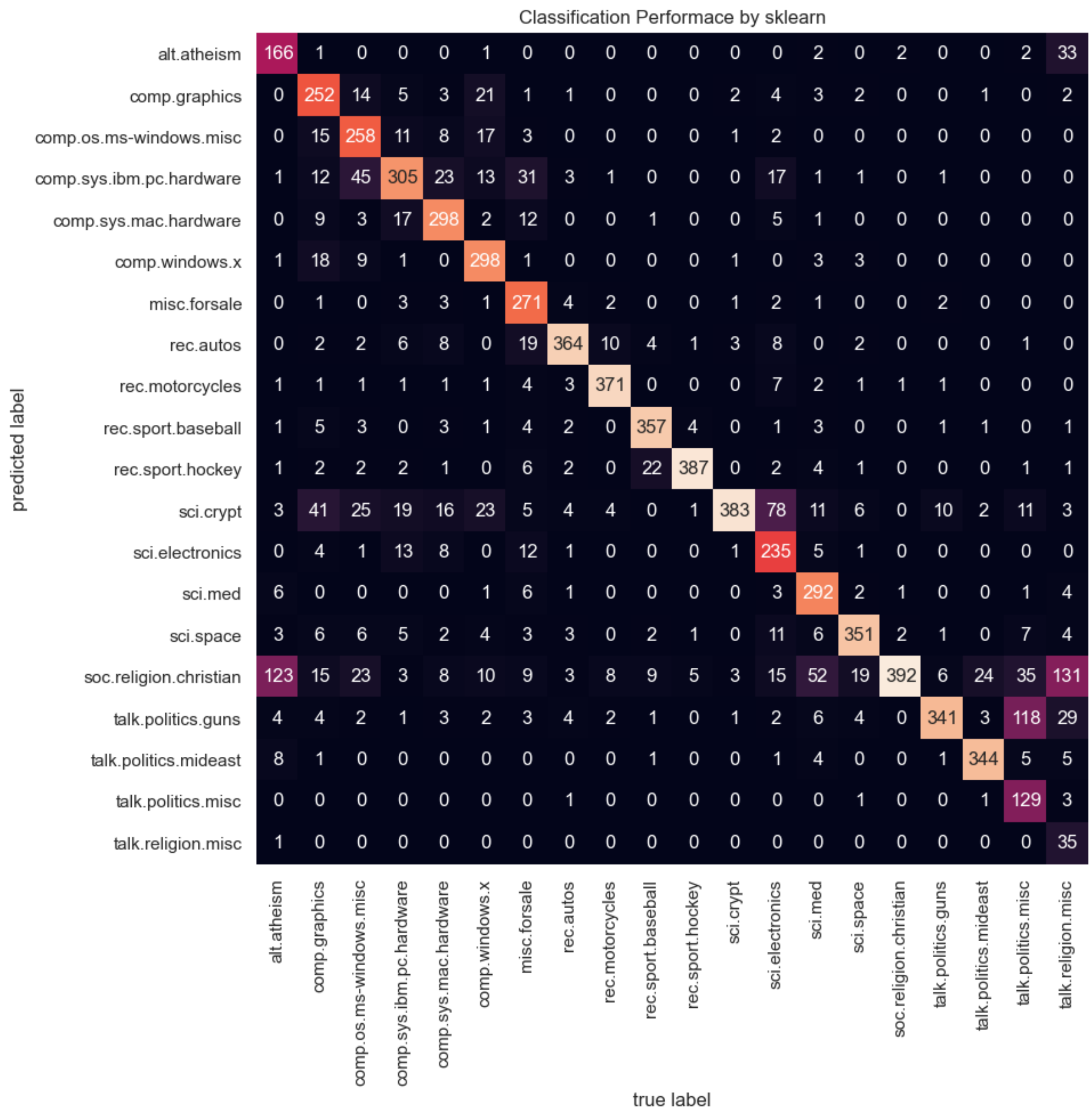
In [6]:

```
# plot classification matrix
mat = confusion_matrix(test_label, predict_test_labels)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False)
plt.title('Classification Performace by my implementation') # fixed a typo here,
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.tight_layout()
plt.savefig('./nb/output/nbm_mine.png')
plt.show()
```




**Reminder:** Do not forget to run nbm_sklearn.py to compare the results to get the accuracy and confusion matrix by sklearn implementation. You can run `python nbm_sklearn.py` under the folder path of `./hw6/nb/` .

`sklearn` results:
Train accuracy: 0.9326
Test accuracy: 0.7734




## Question & Analysis

1. Please indicate whether you implemented based the given code or from scratch.

2. Report your classification accuracy on train and test documents. Also report your classification confusion matrix. Show one example document that Naive Bayes classifies incorrectly (i.e. fill in the following result table). Attach the output figure `./output/nbm_mine.png` in the jupyter book and briefly explain your observation on the accuracy and confusion matrix.

| | Train set accuracy | Test set accuracy |
|---|---|---|
| sklearn implementaion | | |
| your implementaion | | |

1. Show one example document that Naive Bayes classifies incorrectly by filling the following table. Provide your thought on the reason why this document is misclassified. (Note that

the topic mapping is available at `train.map` same as `test.map` )

| Words (count) in the example document | Predicted label | Truth label |
|---|---|---|
| For example, student (4), education (2), ... | Class A | Class B |

1. Is Naive Bayes a generative model or discriminative model and why? What is the difference between Naive Bayes classifier and Logistic Regression? What are the pros and cons of Naive Bayes for text classification task?

2. Can you apply Naive Bayes model to identify spam emails from normal ones? Briefly explain your method (you don't need to implementation for this question).

In [7]:

```
test_res = np.array(test_label) == np.array(predict_test_labels)
misclassified_doc = None
for i, res in enumerate(test_res):
    if not res:
        print(f"A misclassified test document: {i+1}")
        print(f"Predicted {predict_test_labels[i]}, ground truth {test_label[i]}
        misclassified_doc = i + 1
        break

df = df_test.loc[df_test['docIdx'] == misclassified_doc].copy()
vocab_dict = vocab_df.set_index('index').to_dict()
# print(vocab_dict)
df['words'] = df['wordIdx'].map(vocab_df.set_index('index')['word'].to_dict())
print(df)
df.to_csv("document4.csv", sep=",")
```

```
A misclassified test document: 4
Predicted 16, ground truth 1
     docIdx  wordIdx  count               words
395       4       10      1             atheist
396       4       12     20                  of
397       4       16      1                from
398       4       17     10            religion
399       4       23     25                 and
..      ...      ...    ...                 ...
746       4    53976      2                 tgk
747       4    53977      1             quelled
748       4    53978      1      quintessential
749       4    53979      1        philanthropy
750       4    53980      3  supernaturalists

[356 rows x 4 columns]
```

### Your Answers

#### Question 0:

I implemented naive Bayes using the provided code skeleton.

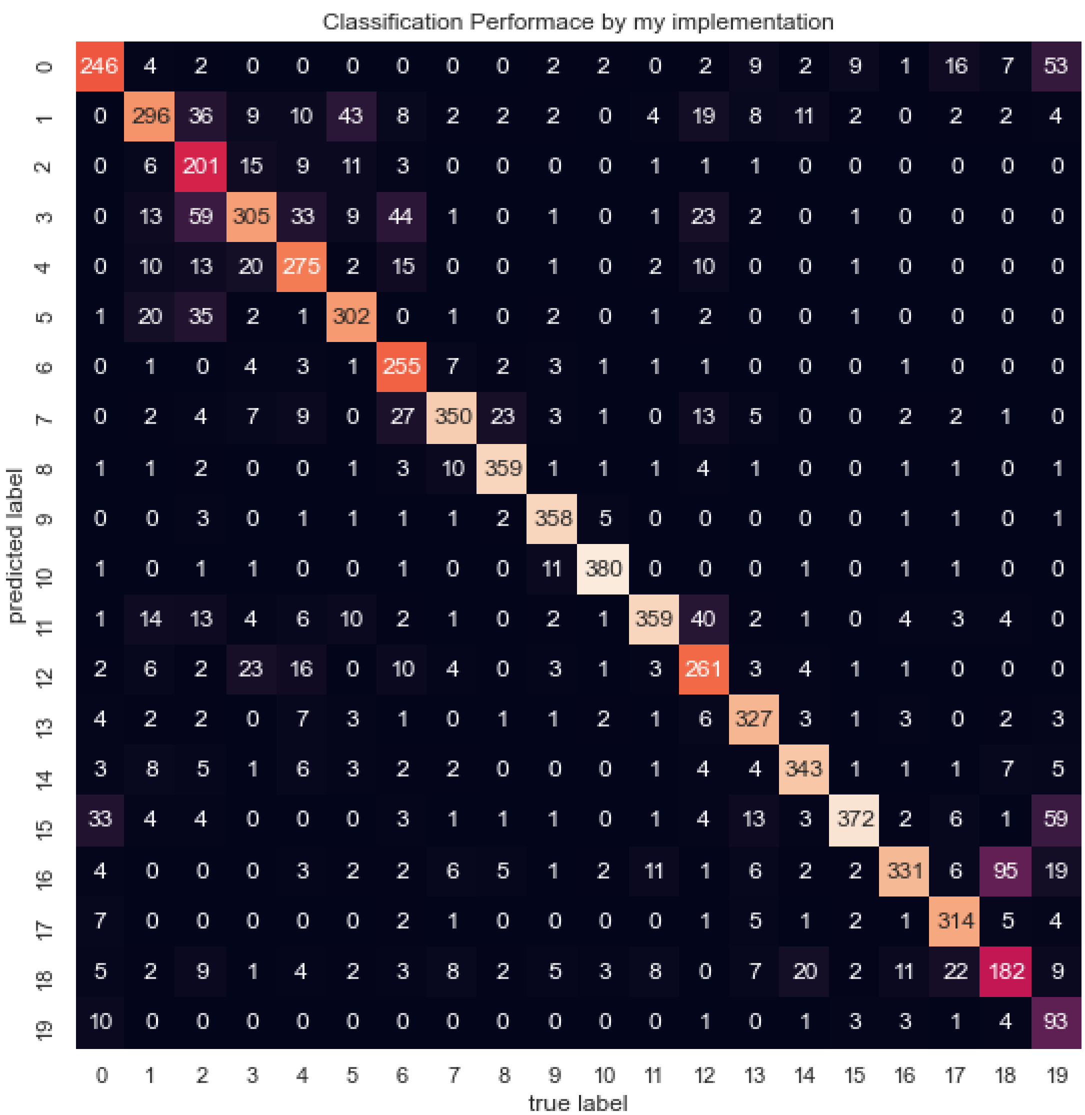
#### Question 1:

My implementation and the `sklearn` implementation had the following train and test set accuracies:

| | Train Set Accuracy | Test Set Accuracy |
|---|---|---|

| | Train Set Accuracy | Test Set Accuracy |
|---|---|---|
| `sklearn` implementation | 0.9326 | 0.7734 |
| Your implementation | 0.9482 | 0.7873 |

The following image gives the classification confusion matrix for my implementation (generated from cell above):




From the confusion matrix, we can see that for most of the classes, what was predicted matched the ground truth. This makes sense, as the accuracy was 0.7873 for test data. However, there were lots of off-diagonal values that were nonzero, which is why the test accuracy is only 0.7873. The notable mispredictions are as follows (all nondiagonal values exceeding 50):

| Ground Truth | Predicted Topic | # of Mispredictions |
|---|---|---|
| talk.politics.misc (column 18/topic 19) | talk.politics.guns (row 16/topic 17) | 95 |
| talk.religion.misc (column 19/topic 20) | soc.religion.christian (row 15/topic 16) | 59 |

| Ground Truth | Predicted Topic | # of Mispredictions |
|---|---|---|
| talk.religion.misc (column 19/topic 20) | alt.atheism (row 0/topic 1) | 53 |
| comp.os.ms-windows.misc (column 2/topic 3) | comp.sys.imb.pc.hardware (row 3/topic 4) | 59 |

Note: the column and row numbers start from 0, so the class/topic number is the row or column number plus 1.

From this table, we can see that for these mispredictions, it tended to be a miscellaneous topic that NB thought actually had a closely related topic. Politics often includes talks about guns, religion includes both Christianity and atheism, and Windows is closely related to IBM PC hardware. Thus, the model was essentially deciding that that various documents in this area classified under "misc" under had a topic.

One particular document that NB misclassified was document 4 from the test set. A portion of the document is printed below. The full document is attached to the end of this submission after the code.

```
A misclassified test document: 4
     docIdx  wordIdx  count            words
395       4       10      1          atheist
396       4       12     20               of
397       4       16      1             from
398       4       17     10         religion
399       4       23     25              and
..      ...      ...    ...              ...
746       4    53976      2              tgk
747       4    53977      1          quelled
748       4    53978      1   quintessential
749       4    53979      1      philanthropy
750       4    53980      3  supernaturalists

[356 rows x 4 columns]
```

**Question 2:**
Document 4 was misclassified. The details about the document are given below.

| Words (count) in document 4 | Predicted label | Truth label |
|---|---|---|
| atheist(1), of(20), from(1), religion(10), and(25)...tgk(2), quelled(1), quintessential(1), philanthropy(1), supernaturalists(3) | Class 16 | Class 1 |

The results indicate that it predicted that document 4 was soc.religion.christian while in reality it was alt.atheism. This misclassification probably occurred because atheism and Christianity are both religion-related topics, so they are somewhat releated. There may have been some words in this atheism document that related to Christianity, and they were enough to make the model think that it was more likely for this document to be one about Christianity. Also, note the word "atheism" only appeared once, so this might have contributed to a lower likelihood for alt.atheism compared to soc.religion.christian. In total, there were 33 documents that were misclassified as class 16 when they were actually class 1, showing that this misclassification isn't that uncommon.

**Question 3:**
Naive Bayes is a generative model because it models a joint probability P(x, y) instead of a conditional probability P(y|x), which corresponds to a discriminative model. In this case, the joint probablity is P(document contains some sequence of words, document is in class j). The difference between NB and logistic regression is that NB assumes that features such as words are conditionally independent given the class label, while logistic regression does not, and thus it learns a conditional probability distribution instead, P(class | some data).

For text classification, the pros of NB are that it is a simple model and runs quite fast, even with large data sets. However, its assumption of conditional independence is not necessarily something that is true in the real world. Also, NB fails to take into account the sequence of words in a document, and so it loses some information by using a bag of words representation.

**Question 4:**
Yes, you could, and this is actually used by many real-world spam detection algorithms. Spam emails commonly contain certain words that immediately cause humans to classify the spam, such as "won," "free," and "offer." My method would be to first train the model on a dataset of emails that includes both spam and non-spam emails so that it can learn the joint probabilities. To classify a new email, I would convert it to a bag words of representation, calculate the likelihood of it being a spam or not a spam email using the learned joint probability parameters, and then make the classification according to which class (spam or not spam) has a higher likelihood from the words in the email.

---

# Section 2: Topic Modeling: Probabilistic Latent Semantic Analysis (50 points)

In this section, you will implement Probabilistic Latent Semantic Analysis (pLSA) by EM algorithm. Note: The code and dataset are under the subfolder named `plsa`. You can find two dataset files named `dataset1.txt` and `dataset2.txt` together with a stopword list as `stopwords.dic`.

First complete the implementation of pLSA in `plsa.py`. You need to finish the E step, M step and likelihood function. Note that the optimizing process on dataset 2 might take a while.

In [6]:
```
# input file, outpot files and parameters
datasetFilePath = './plsa/dataset/dataset1.txt' # or set as './plsa/dataset/data
stopwordsFilePath = './plsa/dataset/stopwords.dic'
docTopicDist = './plsa/output/docTopicDistribution.txt'
topicWordDist = './plsa/output/topicWordDistribution.txt'
dictionary = './plsa/output/dictionary.dic'
topicWords = './plsa/output/topics.txt'

K = 4   # number of topic
```

```
maxIteration = 20 # maxIteration and threshold control the train process
threshold = 3
topicWordsNum = 20 # parameter for output
```

In [7]:

```
from plsa.plsa import PLSA
from plsa.utils import preprocessing
# data processing
N, M, word2id, id2word, X = preprocessing(datasetFilePath, stopwordsFilePath)
```

In [18]:

```
plsa_model = PLSA()
plsa_model.initialize(N, K, M, word2id, id2word, X)

oldLoglikelihood = 1
newLoglikelihood = 1

for i in range(0, maxIteration):
    plsa_model.EStep() #implement E step
    plsa_model.MStep() #implement M step
    newLoglikelihood = plsa_model.LogLikelihood()
    print("[",time.strftime('%Y-%m-%d %H:%M:%S',time.localtime(time.time())),"]"
          "iteration", str(newLoglikelihood))
    # you should see increasing loglikelihood
    if(abs(newLoglikelihood - oldLoglikelihood) < threshold):
        # change to absolute value, or else it terminates after first iteration
        break
    oldLoglikelihood = newLoglikelihood

plsa_model.output(docTopicDist, topicWordDist, dictionary, topicWords, topicWord
```

```
[ 2020-12-11 16:04:59 ] 1 iteration -7605.31997548358
[ 2020-12-11 16:04:59 ] 2 iteration -7448.5138155125205
[ 2020-12-11 16:04:59 ] 3 iteration -7277.3019328973005
[ 2020-12-11 16:04:59 ] 4 iteration -7118.597475533447
[ 2020-12-11 16:04:59 ] 5 iteration -6985.828498139784
[ 2020-12-11 16:04:59 ] 6 iteration -6872.979733331642
[ 2020-12-11 16:04:59 ] 7 iteration -6775.821526663395
[ 2020-12-11 16:04:59 ] 8 iteration -6700.153252936795
[ 2020-12-11 16:04:59 ] 9 iteration -6645.141889223938
[ 2020-12-11 16:04:59 ] 10 iteration -6608.452423000288
[ 2020-12-11 16:04:59 ] 11 iteration -6584.809733086158
[ 2020-12-11 16:05:00 ] 12 iteration -6565.413904979665
[ 2020-12-11 16:05:00 ] 13 iteration -6545.967208851735
[ 2020-12-11 16:05:00 ] 14 iteration -6527.880448716848
[ 2020-12-11 16:05:00 ] 15 iteration -6513.849515115477
[ 2020-12-11 16:05:00 ] 16 iteration -6505.988092755073
[ 2020-12-11 16:05:00 ] 17 iteration -6502.7345270345695
[ 2020-12-11 16:05:00 ] 18 iteration -6501.50946888105
```

In [19]:

```
# Repeat with dataset 2

# input file, outpot files and parameters
datasetFilePath = './plsa/dataset/dataset2.txt'

from plsa.plsa import PLSA
from plsa.utils import preprocessing

N, M, word2id, id2word, X = preprocessing(datasetFilePath, stopwordsFilePath)

plsa_model = PLSA()
```

```
plsa_model.initialize(N, K, M, word2id, id2word, X)

oldLoglikelihood = 1
newLoglikelihood = 1

for i in range(0, maxIteration):
    plsa_model.EStep() #implement E step
    plsa_model.MStep() #implement M step
    newLoglikelihood = plsa_model.LogLikelihood()
    print("[",time.strftime('%Y-%m-%d %H:%M:%S',time.localtime(time.time())),"]"
          "iteration", str(newLoglikelihood))
    # you should see increasing loglikelihood
    if(abs(newLoglikelihood - oldLoglikelihood) < threshold):
        # change to absolute value, or else it terminates after first iteration
        break
    oldLoglikelihood = newLoglikelihood

plsa_model.output(docTopicDist, topicWordDist, dictionary, topicWords, topicWord
```

```
[ 2020-12-11 16:16:58 ] 1 iteration -153054.3762461547
[ 2020-12-11 16:17:01 ] 2 iteration -151543.16479613347
[ 2020-12-11 16:17:05 ] 3 iteration -149823.75824213616
[ 2020-12-11 16:17:08 ] 4 iteration -148058.48170336388
[ 2020-12-11 16:17:12 ] 5 iteration -146476.6381192339
[ 2020-12-11 16:17:15 ] 6 iteration -145186.0633160714
[ 2020-12-11 16:17:18 ] 7 iteration -144163.51895091266
[ 2020-12-11 16:17:22 ] 8 iteration -143369.1620027521
[ 2020-12-11 16:17:25 ] 9 iteration -142768.786576884
[ 2020-12-11 16:17:29 ] 10 iteration -142308.06262680344
[ 2020-12-11 16:17:32 ] 11 iteration -141955.95563714925
[ 2020-12-11 16:17:35 ] 12 iteration -141684.9463008054
[ 2020-12-11 16:17:39 ] 13 iteration -141473.74246865415
[ 2020-12-11 16:17:42 ] 14 iteration -141309.934820173
[ 2020-12-11 16:17:46 ] 15 iteration -141186.01301698684
[ 2020-12-11 16:17:49 ] 16 iteration -141088.47081243386
[ 2020-12-11 16:17:53 ] 17 iteration -141005.52966142842
[ 2020-12-11 16:17:56 ] 18 iteration -140938.28897460527
[ 2020-12-11 16:17:59 ] 19 iteration -140891.1357450914
[ 2020-12-11 16:18:03 ] 20 iteration -140855.5766698348
```

#### Question & Analysis

1. Please indicate whether you implemented based the given code or from scratch.
2. Choose different $K$ (number of topics) in `plsa.py` . What is your option for a reasonable $K$ in `dataset1.txt` and `dataset2.txt` ? Give your results of 10 words under each topic by filling in the following table (suppose you set $K = 4$).

For dataset 1:

| **Topic 1** | **Topic 2** | **Topic 3** | **Topic 4** |
|---|---|---|---|
| *your words* | *your words* | *your words* | *your words* |

For dataset 2:

| **Topic 1** | **Topic 2** | **Topic 3** | **Topic 4** |
|---|---|---|---|
| *your words* | *your words* | *your words* | *your words* |

1. Are there any similarities between pLSA and GMM model? Briefly explain your thoughts.

2. What are the disadvantages of pLSA? Consider its generalizing ability to new unseen document and its parameter complexity, etc.

**Answers are given below these code blocks.**

In [20]:

```
# Try dataset1 with different K values
datasetFilePath = './plsa/dataset/dataset1.txt'

from plsa.plsa import PLSA
from plsa.utils import preprocessing

N, M, word2id, id2word, X = preprocessing(datasetFilePath, stopwordsFilePath)

for K in [2, 3, 4, 5]:
    plsa_model = PLSA()
    plsa_model.initialize(N, K, M, word2id, id2word, X)

    oldLoglikelihood = 1
    newLoglikelihood = 1

    for i in range(0, maxIteration):
        plsa_model.EStep() #implement E step
        plsa_model.MStep() #implement M step
        newLoglikelihood = plsa_model.LogLikelihood()
        if(abs(newLoglikelihood - oldLoglikelihood) < threshold):
            # change to absolute value, or else it terminates after first iterat
            break
        oldLoglikelihood = newLoglikelihood
    print(f"[{time.strftime('%Y-%m-%d %H:%M:%S',time.localtime(time.time()))}] "
          f"Log likelihood for K={K}: {oldLoglikelihood}")

    plsa_model.output(docTopicDist, topicWordDist, dictionary, topicWords, topic

    i = 1
    print(f"| Topic | Words |\n|-------|-------|")
    with open(topicWords, 'r') as f:
        for line in f:
            words = line.split(' ')
            non_punct_words = []
            for w in words:
                if len(non_punct_words) == 10:
                    break
                if w.isalpha(): # exclude '' and ``
                    non_punct_words.append(w)

            print(f"|{i} | {non_punct_words}")
            i += 1
```

```
[2020-12-11 16:20:02] Log likelihood for K=2: -7412.032040954314
| Topic | Words |
|-------|-------|
|1 | ['sea', 'piece', 'devil', 'pirates', 'fruit', 'grand', 'luffy', 'user', 'ha
ki', 'blue']
|2 | ['luffy', 'crew', 'pirates', 'island', 'straw', 'hat', 'franky', 'dressros
a', 'government', 'robin']
[2020-12-11 16:20:03] Log likelihood for K=3: -7023.921855293514
| Topic | Words |
|-------|-------|
|1 | ['luffy', 'pirates', 'grand', 'roger', 'sea', 'navy', 'crew', 'haki', 'piec
e', 'blue']
```

```
|2 | ['devil', 'fruit', 'sea', 'manga', 'user', 'fruits', 'island', 'piece', 'lu
ffy', 'baroque']
|3 | ['crew', 'luffy', 'island', 'pirates', 'straw', 'dressrosa', 'alliance', 'f
ranky', 'battle', 'government']
[2020-12-11 16:20:04] Log likelihood for K=4: -6697.731847739007
| Topic | Words |
|-------|-------|
|1 | ['luffy', 'pirates', 'island', 'dressrosa', 'crew', 'straw', 'alliance', 'n
ami', 'fishman', 'captain']
|2 | ['crew', 'luffy', 'pirates', 'island', 'piece', 'manga', 'government', 'gra
nd', 'straw', 'war']
|3 | ['sea', 'devil', 'fruit', 'grand', 'user', 'fruits', 'called', 'water', 'mo
untain', 'burū']
|4 | ['haki', 'pirates', 'zou', 'baroque', 'alabasta', 'luffy', 'color', 'islan
d', 'jack', 'zunisha']
[2020-12-11 16:20:05] Log likelihood for K=5: -6550.837091607988
| Topic | Words |
|-------|-------|
|1 | ['luffy', 'dressrosa', 'ace', 'crew', 'alliance', 'navy', 'pirate', 'serie
s', 'doflamingo', 'defeat']
|2 | ['haki', 'grand', 'sea', 'government', 'red', 'blue', 'mountain', 'burū',
'animals', 'color']
|3 | ['devil', 'fruit', 'pirates', 'crew', 'luffy', 'user', 'fruits', 'sea', 'st
raw', 'alabasta']
|4 | ['luffy', 'pirates', 'manga', 'island', 'piece', 'fishman', 'crew', 'captai
n', 'arlong', 'nami']
|5 | ['island', 'zou', 'pose', 'magnetic', 'log', 'set', 'mom', 'charlotte', 'pi
rates', 'luffy']
```

In [21]:

```python
# Repeat with dataset 2

# input file, outpot files and parameters
datasetFilePath = './plsa/dataset/dataset2.txt'

from plsa.plsa import PLSA
from plsa.utils import preprocessing

N, M, word2id, id2word, X = preprocessing(datasetFilePath, stopwordsFilePath)

# Try dataset1 with different K values
for K in [10, 20, 30]:
    plsa_model = PLSA()
    plsa_model.initialize(N, K, M, word2id, id2word, X)

    oldLoglikelihood = 1
    newLoglikelihood = 1

    for i in range(0, maxIteration):
        print(f"{i},", end="", flush=True)
        plsa_model.EStep() #implement E step
        plsa_model.MStep() #implement M step
        newLoglikelihood = plsa_model.LogLikelihood()
        if(abs(newLoglikelihood - oldLoglikelihood) < threshold):
            # change to absolute value, or else it terminates after first iterat
            break
        oldLoglikelihood = newLoglikelihood
    print(f"\n[{time.strftime('%Y-%m-%d %H:%M:%S',time.localtime(time.time()))}]
          f"Log likelihood for K={K}: {oldLoglikelihood}")

    plsa_model.output(docTopicDist, topicWordDist, dictionary, topicWords, topic

    i = 1
```

```
    print(f"| Topic | Words |\n|-------|-------|")
    with open(topicWords, 'r') as f:
        for line in f:
            words = line.split(' ')
            non_punct_words = []
            for w in words:
                if len(non_punct_words) == 10:
                    break
                if w.isalpha():
                    non_punct_words.append(w)

            print(f"|{i} | {non_punct_words}")
            i += 1
```

```
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,
[2020-12-11 16:34:44] Log likelihood for K=10: -128149.27220865787
| Topic | Words |
|-------|-------|
|1 | ['soviet', 'bush', 'fire', 'gorbachev', 'president', 'mexico', 'pope', 'mil
itary', 'church', 'officials']
|2 | ['noriega', 'dukakis', 'jackson', 'california', 'officials', 'president',
'economic', 'panama', 'north', 'kim']
|3 | ['rating', 'greyhound', 'gorbachev', 'warming', 'union', 'global', 'study',
'leaders', 'officials', 'percent']
|4 | ['central', 'snow', 'fbi', 'season', 'agents', 'northern', 'southern', 'exp
ected', 'heavy', 'inches']
|5 | ['roberts', 'magellan', 'people', 'spacecraft', 'city', 'contact', 'polic
e', 'national', 'pictures', 'trible']
|6 | ['barry', 'union', 'soviet', 'officers', 'moore', 'polish', 'friday', 'plan
t', 'immigration', 'people']
|7 | ['bush', 'campaign', 'oil', 'police', 'waste', 'peres', 'company', 'flori
o', 'people', 'exxon']
|8 | ['administration', 'farmer', 'police', 'people', 'settlements', 'school',
'receptor', 'occupied', 'government', 'shamir']
|9 | ['bank', 'company', 'president', 'central', 'time', 'children', 'congress',
'nikolais', 'germany', 'jews']
|10 | ['percent', 'prices', 'rate', 'report', 'rose', 'business', 'economy', 'du
racell', 'month', 'billion']
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,
[2020-12-11 16:35:56] Log likelihood for K=20: -118445.30428615403
| Topic | Words |
|-------|-------|
|1 | ['kim', 'north', 'receptor', 'rose', 'roh', 'businesses', 'scientists', 'op
position', 'korean', 'visit']
|2 | ['people', 'immigration', 'administration', 'farmer', 'farm', 'percent', 'c
ity', 'nelson', 'enforcement', 'border']
|3 | ['company', 'bar', 'multistate', 'ferrets', 'doctors', 'percent', 'question
s', 'stock', 'suit', 'dorrance']
|4 | ['school', 'pope', 'church', 'boy', 'mexico', 'animals', 'teacher', 'peopl
e', 'police', 'salinas']
|5 | ['roberts', 'greyhound', 'people', 'drivers', 'union', 'company', 'injure
d', 'fear', 'bombs', 'violence']
|6 | ['bush', 'union', 'president', 'campaign', 'people', 'batalla', 'cordero',
'maxwell', 'societe', 'generale']
|7 | ['bank', 'fire', 'duracell', 'monday', 'billion', 'company', 'kraft', 'summ
er', 'record', 'sales']
|8 | ['waste', 'police', 'county', 'orr', 'people', 'officers', 'nelson', 'sit
e', 'threat', 'friday']
|9 | ['rating', 'percent', 'films', 'mpaa', 'military', 'combe', 'system', 'pres
ident', 'government', 'inventories']
|10 | ['noriega', 'magellan', 'spacecraft', 'panama', 'jackson', 'contact', 'sau
di', 'american', 'forces', 'pictures']
|11 | ['soviet', 'gorbachev', 'florio', 'polish', 'study', 'plant', 'exxon', 'of
ficers', 'arco', 'union']
```

```
|12 | ['grain', 'iraq', 'baker', 'iran', 'saudi', 'united', 'guerrillas', 'syri
a', 'warhol', 'zennoh']
|13 | ['central', 'snow', 'embassy', 'washington', 'northern', 'wednesday', 'sou
thern', 'inches', 'nation', 'degrees']
|14 | ['percent', 'oil', 'prices', 'gas', 'inflation', 'rate', 'skins', 'septemb
er', 'thursday', 'enron']
|15 | ['percent', 'soviet', 'fbi', 'agents', 'production', 'november', 'octobe
r', 'trible', 'rate', 'senate']
|16 | ['dukakis', 'elephant', 'jackson', 'national', 'told', 'roberson', 'polic
e', 'front', 'car', 'zoo']
|17 | ['bush', 'gorbachev', 'soviet', 'museum', 'liberace', 'police', 'dukakis',
'bloomberg', 'top', 'president']
|18 | ['warming', 'global', 'children', 'settlements', 'summit', 'nikolais', 'sh
amir', 'occupied', 'environmental', 'forestry']
|19 | ['congress', 'government', 'jews', 'jewish', 'germany', 'berlin', 'peopl
e', 'waldheim', 'officials', 'east']
|20 | ['barry', 'california', 'peres', 'moore', 'official', 'rappaport', 'bechte
l', 'offer', 'israel', 'mundy']
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,
[2020-12-11 16:37:18] Log likelihood for K=30: -111812.88282304422
| Topic | Words |
|-------|-------|
|1 | ['fire', 'immigration', 'nelson', 'border', 'monday', 'fiscal', 'firefighte
rs', 'forest', 'officials', 'biaggi']
|2 | ['ago', 'president', 'history', 'american', 'died', 'york', 'time', 'minist
er', 'star', 'organization']
|3 | ['warming', 'global', 'summit', 'receptor', 'scientists', 'environmental',
'gene', 'brain', 'forestry', 'leaders']
|4 | ['farmer', 'soviet', 'administration', 'polish', 'people', 'officers', 'uni
on', 'farm', 'percent', 'farmers']
|5 | ['plant', 'arco', 'employees', 'chemical', 'people', 'investigation', 'kill
ed', 'sorgenti', 'workers', 'pounds']
|6 | ['congress', 'jews', 'germany', 'berlin', 'jewish', 'waldheim', 'meeting',
'people', 'israel', 'garbo']
|7 | ['bush', 'dukakis', 'prosperity', 'night', 'peace', 'president', 'campaig
n', 'trade', 'earlier', 'day']
|8 | ['saudi', 'iraq', 'gorbachev', 'forces', 'soviet', 'arabia', 'iran', 'bake
r', 'tanks', 'united']
|9 | ['gorbachev', 'magellan', 'spacecraft', 'study', 'economy', 'soviet', 'cont
act', 'economic', 'pictures', 'earth']
|10 | ['peres', 'official', 'offer', 'rappaport', 'bechtel', 'israel', 'robb',
'oil', 'governor', 'memo']
|11 | ['roberts', 'greyhound', 'union', 'drivers', 'people', 'company', 'negotia
tions', 'violence', 'fear', 'officials']
|12 | ['north', 'kim', 'talks', 'government', 'leader', 'roh', 'visit', 'korea
n', 'opposition', 'rebel']
|13 | ['soviet', 'immigration', 'calgary', 'combe', 'anthrax', 'week', 'mireck
i', 'romanian', 'coach', 'biological']
|14 | ['fbi', 'agents', 'children', 'nikolais', 'bar', 'multistate', 'suit', 'hi
spanic', 'questions', 'program']
|15 | ['embassy', 'national', 'cuban', 'government', 'diplomatic', 'czechoslova
k', 'doctors', 'ferrets', 'people', 'washington']
|16 | ['bank', 'record', 'summer', 'england', 'monday', 'power', 'maxwell', 'bil
lion', 'plan', 'generale']
|17 | ['percent', 'prices', 'inventories', 'july', 'average', 'survey', 'held',
'fell', 'stock', 'month']
|18 | ['rose', 'businesses', 'percent', 'animals', 'elephant', 'soviet', 'firm
s', 'zoo', 'committee', 'europe']
|19 | ['bombs', 'roberson', 'car', 'black', 'killing', 'burt', 'biggest', 'japan
ese', 'injured', 'rest']
|20 | ['waste', 'police', 'liberace', 'grain', 'official', 'trible', 'bloomber
g', 'museum', 'orr', 'money']
|21 | ['percent', 'california', 'gas', 'oil', 'rate', 'production', 'rose', 'fed
eral', 'november', 'air']
|22 | ['noriega', 'panama', 'settlements', 'jackson', 'israel', 'shamir', 'occup
```

```
ied', 'powell', 'letter', 'committee']
|23 | ['central', 'snow', 'degrees', 'southern', 'northern', 'temperatures', 'in
ches', 'nation', 'wednesday', 'expected']
|24 | ['bush', 'campaign', 'school', 'boy', 'police', 'teacher', 'students', 'fa
mily', 'shot', 'president']
|25 | ['rating', 'system', 'skins', 'films', 'mpaa', 'city', 'japan', 'tie', 'sh
elter', 'mcguire']
|26 | ['pope', 'church', 'mexico', 'salinas', 'people', 'visit', 'constitution',
'government', 'president', 'hurricane']
|27 | ['dukakis', 'florio', 'exxon', 'jackson', 'programs', 'democratic', 'unio
n', 'spill', 'pittsburgh', 'drug']
|28 | ['company', 'duracell', 'kraft', 'kravis', 'percent', 'kohlberg', 'billio
n', 'financial', 'sales', 'close']
|29 | ['season', 'ratings', 'nbc', 'tuesday', 'abc', 'cosby', 'percent', 'serie
s', 'cbs', 'wo']
|30 | ['percent', 'barry', 'prices', 'moore', 'inflation', 'oil', 'batalla', 'mu
ndy', 'mayor', 'september']
```

### Your Answers

#### Question 0:

I implemented PLSA based on the skeleton code.

#### Question 1:

For `dataset1.txt` , values of K between 2-5 seem to be reasonable, as some of the documents talk about the background of One Piece, while other documents describe the plot or setting of One Piece. 3 may be an especially well suited number because I see roughly 3 categories of documents: some history/background/reviews about One Piece, then descriptions about its setting, and then its plot. The results are given below:

K=2:

| Topic | Words |
|---|---|
| 1 | ['sea', 'piece', 'devil', 'pirates', 'fruit', 'grand', 'luffy', 'user', 'haki', 'blue'] |
| 2 | ['luffy', 'crew', 'pirates', 'island', 'straw', 'hat', 'franky', 'dressrosa', 'government', 'robin'] |

K=3:

| Topic | Words |
|---|---|
| 1 | ['luffy', 'pirates', 'grand', 'roger', 'sea', 'navy', 'crew', 'haki', 'piece', 'blue'] |
| 2 | ['devil', 'fruit', 'sea', 'manga', 'user', 'fruits', 'island', 'piece', 'luffy', 'baroque'] |
| 3 | ['crew', 'luffy', 'island', 'pirates', 'straw', 'dressrosa', 'alliance', 'franky', 'battle', 'government'] |

K=4:

| Topic | Words |
|---|---|
| 1 | ['luffy', 'pirates', 'island', 'dressrosa', 'crew', 'straw', 'alliance', 'nami', 'fishman', 'captain'] |
| 2 | ['crew', 'luffy', 'pirates', 'island', 'piece', 'manga', 'government', 'grand', 'straw', 'war'] |
| 3 | ['sea', 'devil', 'fruit', 'grand', 'user', 'fruits', 'called', 'water', 'mountain', 'burū'] |
| 4 | ['haki', 'pirates', 'zou', 'baroque', 'alabasta', 'luffy', 'color', 'island', 'jack', 'zunisha'] |

K=5:

| Topic | Words |
|---|---|
| 1 | ['luffy', 'dressrosa', 'ace', 'crew', 'alliance', 'navy', 'pirate', 'series', 'doflamingo', 'defeat'] |
| 2 | ['haki', 'grand', 'sea', 'government', 'red', 'blue', 'mountain', 'burū', 'animals', 'color'] |
| 3 | ['devil', 'fruit', 'pirates', 'crew', 'luffy', 'user', 'fruits', 'sea', 'straw', 'alabasta'] |
| 4 | ['luffy', 'pirates', 'manga', 'island', 'piece', 'fishman', 'crew', 'captain', 'arlong', 'nami'] |
| 5 | ['island', 'zou', 'pose', 'magnetic', 'log', 'set', 'mom', 'charlotte', 'pirates', 'luffy'] |

For all values of K, it seems like the topics are somewhat different, making these values of K reasonable.

For `dataset2.txt` , using the same procedure, it seems like K=10, 20, or 30 all work fairly well. The first 10 words in each topic relate to each other. As K increases, the topics become more specific, which makes sense, as increased K means more classes, so pLSA finds finer divisions between the set of documents. As the topics get more specific, it becomes easier to see that the 10 words for that class are all related in some way. The results are given below:

K=10:

| Topic | Words |
|---|---|
| 1 | ['soviet', 'bush', 'fire', 'gorbachev', 'president', 'mexico', 'pope', 'military', 'church', 'officials'] |
| 2 | ['noriega', 'dukakis', 'jackson', 'california', 'officials', 'president', 'economic', 'panama', 'north', 'kim'] |
| 3 | ['rating', 'greyhound', 'gorbachev', 'warming', 'union', 'global', 'study', 'leaders', 'officials', 'percent'] |
| 4 | ['central', 'snow', 'fbi', 'season', 'agents', 'northern', 'southern', 'expected', 'heavy', 'inches'] |
| 5 | ['roberts', 'magellan', 'people', 'spacecraft', 'city', 'contact', 'police', 'national', 'pictures', 'trible'] |
| 6 | ['barry', 'union', 'soviet', 'officers', 'moore', 'polish', 'friday', 'plant', 'immigration', 'people'] |
| 7 | ['bush', 'campaign', 'oil', 'police', 'waste', 'peres', 'company', 'florio', 'people', 'exxon'] |
| 8 | ['administration', 'farmer', 'police', 'people', 'settlements', 'school', 'receptor', 'occupied', 'government', 'shamir'] |
| 9 | ['bank', 'company', 'president', 'central', 'time', 'children', 'congress', 'nikolais', 'germany', 'jews'] |
| 10 | ['percent', 'prices', 'rate', 'report', 'rose', 'business', 'economy', 'duracell', 'month', 'billion'] |

K=20:

| Topic | Words |
|---|---|
| 1 | ['kim', 'north', 'receptor', 'rose', 'roh', 'businesses', 'scientists', 'opposition', 'korean', 'visit'] |
| 2 | ['people', 'immigration', 'administration', 'farmer', 'farm', 'percent', 'city', 'nelson', 'enforcement', 'border'] |
| 3 | ['company', 'bar', 'multistate', 'ferrets', 'doctors', 'percent', 'questions', 'stock', 'suit', 'dorrance'] |
| 4 | ['school', 'pope', 'church', 'boy', 'mexico', 'animals', 'teacher', 'people', 'police', 'salinas'] |
| 5 | ['roberts', 'greyhound', 'people', 'drivers', 'union', 'company', 'injured', 'fear', 'bombs', 'violence'] |

| Topic | Words |
|---|---|
| 6 | ['bush', 'union', 'president', 'campaign', 'people', 'batalla', 'cordero', 'maxwell', 'societe', 'generale'] |
| 7 | ['bank', 'fire', 'duracell', 'monday', 'billion', 'company', 'kraft', 'summer', 'record', 'sales'] |
| 8 | ['waste', 'police', 'county', 'orr', 'people', 'officers', 'nelson', 'site', 'threat', 'friday'] |
| 9 | ['rating', 'percent', 'films', 'mpaa', 'military', 'combe', 'system', 'president', 'government', 'inventories'] |
| 10 | ['noriega', 'magellan', 'spacecraft', 'panama', 'jackson', 'contact', 'saudi', 'american', 'forces', 'pictures'] |
| 11 | ['soviet', 'gorbachev', 'florio', 'polish', 'study', 'plant', 'exxon', 'officers', 'arco', 'union'] |
| 12 | ['grain', 'iraq', 'baker', 'iran', 'saudi', 'united', 'guerrillas', 'syria', 'warhol', 'zennoh'] |
| 13 | ['central', 'snow', 'embassy', 'washington', 'northern', 'wednesday', 'southern', 'inches', 'nation', 'degrees'] |
| 14 | ['percent', 'oil', 'prices', 'gas', 'inflation', 'rate', 'skins', 'september', 'thursday', 'enron'] |
| 15 | ['percent', 'soviet', 'fbi', 'agents', 'production', 'november', 'october', 'trible', 'rate', 'senate'] |
| 16 | ['dukakis', 'elephant', 'jackson', 'national', 'told', 'roberson', 'police', 'front', 'car', 'zoo'] |
| 17 | ['bush', 'gorbachev', 'soviet', 'museum', 'liberace', 'police', 'dukakis', 'bloomberg', 'top', 'president'] |
| 18 | ['warming', 'global', 'children', 'settlements', 'summit', 'nikolais', 'shamir', 'occupied', 'environmental', 'forestry'] |
| 19 | ['congress', 'government', 'jews', 'jewish', 'germany', 'berlin', 'people', 'waldheim', 'officials', 'east'] |
| 20 | ['barry', 'california', 'peres', 'moore', 'official', 'rappaport', 'bechtel', 'offer', 'israel', 'mundy'] |

K=30:

| Topic | Words |
|---|---|
| 1 | ['fire', 'immigration', 'nelson', 'border', 'monday', 'fiscal', 'firefighters', 'forest', 'officials', 'biaggi'] |
| 2 | ['ago', 'president', 'history', 'american', 'died', 'york', 'time', 'minister', 'star', 'organization'] |
| 3 | ['warming', 'global', 'summit', 'receptor', 'scientists', 'environmental', 'gene', 'brain', 'forestry', 'leaders'] |
| 4 | ['farmer', 'soviet', 'administration', 'polish', 'people', 'officers', 'union', 'farm', 'percent', 'farmers'] |
| 5 | ['plant', 'arco', 'employees', 'chemical', 'people', 'investigation', 'killed', 'sorgenti', 'workers', 'pounds'] |
| 6 | ['congress', 'jews', 'germany', 'berlin', 'jewish', 'waldheim', 'meeting', 'people', 'israel', 'garbo'] |
| 7 | ['bush', 'dukakis', 'prosperity', 'night', 'peace', 'president', 'campaign', 'trade', 'earlier', 'day'] |
| 8 | ['saudi', 'iraq', 'gorbachev', 'forces', 'soviet', 'arabia', 'iran', 'baker', 'tanks', 'united'] |
| 9 | ['gorbachev', 'magellan', 'spacecraft', 'study', 'economy', 'soviet', 'contact', 'economic', 'pictures', 'earth'] |
| 10 | ['peres', 'official', 'offer', 'rappaport', 'bechtel', 'israel', 'robb', 'oil', 'governor', 'memo'] |
| 11 | ['roberts', 'greyhound', 'union', 'drivers', 'people', 'company', 'negotiations', 'violence', 'fear', 'officials'] |
| 12 | ['north', 'kim', 'talks', 'government', 'leader', 'roh', 'visit', 'korean', 'opposition', 'rebel'] |
| 13 | ['soviet', 'immigration', 'calgary', 'combe', 'anthrax', 'week', 'mirecki', 'romanian', 'coach', 'biological'] |
| 14 | ['fbi', 'agents', 'children', 'nikolais', 'bar', 'multistate', 'suit', 'hispanic', 'questions', 'program'] |

| Topic | Words |
|---|---|
| 15 | ['embassy', 'national', 'cuban', 'government', 'diplomatic', 'czechoslovak', 'doctors', 'ferrets', 'people', 'washington'] |
| 16 | ['bank', 'record', 'summer', 'england', 'monday', 'power', 'maxwell', 'billion', 'plan', 'generale'] |
| 17 | ['percent', 'prices', 'inventories', 'july', 'average', 'survey', 'held', 'fell', 'stock', 'month'] |
| 18 | ['rose', 'businesses', 'percent', 'animals', 'elephant', 'soviet', 'firms', 'zoo', 'committee', 'europe'] |
| 19 | ['bombs', 'roberson', 'car', 'black', 'killing', 'burt', 'biggest', 'japanese', 'injured', 'rest'] |
| 20 | ['waste', 'police', 'liberace', 'grain', 'official', 'trible', 'bloomberg', 'museum', 'orr', 'money'] |
| 21 | ['percent', 'california', 'gas', 'oil', 'rate', 'production', 'rose', 'federal', 'november', 'air'] |
| 22 | ['noriega', 'panama', 'settlements', 'jackson', 'israel', 'shamir', 'occupied', 'powell', 'letter', 'committee'] |
| 23 | ['central', 'snow', 'degrees', 'southern', 'northern', 'temperatures', 'inches', 'nation', 'wednesday', 'expected'] |
| 24 | ['bush', 'campaign', 'school', 'boy', 'police', 'teacher', 'students', 'family', 'shot', 'president'] |
| 25 | ['rating', 'system', 'skins', 'films', 'mpaa', 'city', 'japan', 'tie', 'shelter', 'mcguire'] |
| 26 | ['pope', 'church', 'mexico', 'salinas', 'people', 'visit', 'constitution', 'government', 'president', 'hurricane'] |
| 27 | ['dukakis', 'florio', 'exxon', 'jackson', 'programs', 'democratic', 'union', 'spill', 'pittsburgh', 'drug'] |
| 28 | ['company', 'duracell', 'kraft', 'kravis', 'percent', 'kohlberg', 'billion', 'financial', 'sales', 'close'] |
| 29 | ['season', 'ratings', 'nbc', 'tuesday', 'abc', 'cosby', 'percent', 'series', 'cbs', 'wo'] |
| 30 | ['percent', 'barry', 'prices', 'moore', 'inflation', 'oil', 'batalla', 'mundy', 'mayor', 'september'] |

**Question 2:**

There are indeed similarities between pLSA and GMM. Both are probabilistic clustering models that aim to find clusters present in unlabeled data. Instead of being a hard assignment of data points or text to a cluster, they return a probability instead. Both use the expectation-maximization algorithm to arrive at a local minimum for log-likelihood (loss). The main difference is that pSLA is for clustering text data, while GMM is for clustering vector data. Also, GMM assumes a Gaussian distribution, while pLSA assumes a categorical or multinomial distribution.

**Question 3:**

One disadvantage is a slow runtime: `dataset2.txt` isn't that big of a text file (only ~750 KB) but the algorithm takes an extremely long time to run. One has to iterate over all the documents, all the clusters, and all the words in the vocabulary to calculate log likelihood and the model parameters. Also, like GMM, there is no guarantee that pLSA will find a global optimum; it will only find a local optimum. Anoher con is you need to specify the value of K before training the model. It also relies on the assumption of conditional independence, like naive Bayes, which may not hold in the real world. Finally, it has a large number of parameters ($P(z|d)$, $P(w|z)$, and $P(z|d, w)$ are 2D or 3D matrices), and so tuning all of these takes quite some time (see the slow runtime point above). This makes it easy for pLSA to overfit. pLSA also doesn't have a well-defined generative model, so it does poorly in generalizing to new documents that it wasn't trained on.

# Bonus Questions (10 points): LDA

We've learned document and topic modeling techiques. As mentioned in the lecture, most frequently used topic models are pLSA and LDA. Latent Dirichlet allocation (LDA) proposed by David M. Blei, Andrew Y. Ng, and Michael I. Jordan, posits that each document is generated as a mixture of topics where the continuous-valued mixture proportions are distributed as a latent Dirichlet random variable.

In this question, please read the paper and/or tutorials of LDA and finish the following questions and tasks:

(1) What are the differences between pLSA and LDA? List at least one advantage of LDA over pLSA?

(2) Show a demo of LDA with brief result analysis on any corpus and discuss what real-world applications can be supported by LDA. Note: You do not need to implement LDA algorithms from scratch. You may use multiple packages such as `nltk`, `gensim`, `pyLDAvis` (added on the `cs145hw6.yml`) to help show the demo within couple of lines of code. If you'd like to use other packages, feel free to install them.

In [23]:

```
# NOTE: '' and `` were added to stopwords.dic so that they
#       wouldn't be included as words in the processed documents.
import nltk
import gensim
from gensim.models.ldamodel import LdaModel
from plsa.plsa import PLSA
from plsa.utils import preprocessing

# input file, output files and parameters
K = 10
datasetFilePath = './plsa/dataset/dataset2.txt'
N, M, word2id, id2word, X = preprocessing(datasetFilePath, stopwordsFilePath) #

common_corpus = []
for doc in X:
    common_corpus.append([])
    for i, count in enumerate(doc):
        common_corpus[-1].append((i, count))

lda = LdaModel(common_corpus, num_topics=K, id2word=id2word)
print(lda)

for t in lda.show_topics(num_topics=K, num_words=20):
    print(f"Topic {t[0]}:")
    tuples = t[1].split(' + ')
    for t in tuples:
        word_prob = t.split('*')
        print(f"[{word_prob[1][1:-1]}, {word_prob[0]}] ", end="")
    print("\n")
```

```
LdaModel(num_terms=6451, num_topics=10, decay=0.5, chunksize=2000)
Topic 0:
[u.s., 0.004] [percent, 0.004] [soviet, 0.003] [people, 0.003] [government, 0.00
3] [company, 0.003] [officials, 0.003] [police, 0.002] [magellan, 0.002] [dog,
0.002] [president, 0.002] [spacecraft, 0.002] [york, 0.002] [economic, 0.002] [s
```

cientists, 0.002] [washington, 0.002] [california, 0.002] [wednesday, 0.002] [ex
pected, 0.002] [bush, 0.002]

Topic 1:
[president, 0.003] [plant, 0.003] [monday, 0.002] [arco, 0.002] [federal, 0.002]
[officials, 0.002] [california, 0.002] [people, 0.002] [york, 0.002] [percent,
0.002] [school, 0.002] [air, 0.002] [employees, 0.002] [u.s., 0.002] [rose, 0.00
2] [tuesday, 0.002] [county, 0.002] [saudi, 0.002] [warming, 0.002] [enforcemen
t, 0.002]

Topic 2:
[police, 0.005] [bank, 0.004] [mrs., 0.004] [children, 0.003] [soviet, 0.003] [o
fficials, 0.003] [percent, 0.003] [ferrets, 0.002] [doctors, 0.002] [people, 0.0
02] [bloomberg, 0.002] [museum, 0.002] [gorbachev, 0.002] [roberts, 0.002] [cent
er, 0.002] [union, 0.002] [city, 0.002] [national, 0.002] [liberace, 0.002] [kil
led, 0.002]

Topic 3:
[people, 0.006] [president, 0.004] [percent, 0.004] [soviet, 0.003] [bush, 0.00
3] [government, 0.003] [u.s., 0.003] [officials, 0.002] [city, 0.002] [military,
0.002] [administration, 0.002] [gorbachev, 0.002] [dukakis, 0.002] [time, 0.002]
[prices, 0.002] [union, 0.002] [told, 0.002] [company, 0.002] [reported, 0.002]
[national, 0.002]

Topic 4:
[percent, 0.007] [u.s., 0.004] [oil, 0.004] [people, 0.003] [official, 0.003] [s
oviet, 0.003] [union, 0.002] [government, 0.002] [wednesday, 0.002] [day, 0.002]
[central, 0.002] [company, 0.002] [prices, 0.002] [degrees, 0.002] [rate, 0.002]
[monday, 0.002] [peres, 0.002] [war, 0.002] [gas, 0.002] [northern, 0.002]

Topic 5:
[company, 0.005] [duracell, 0.004] [people, 0.004] [percent, 0.004] [bush, 0.00
4] [president, 0.003] [government, 0.003] [official, 0.003] [rating, 0.003] [thu
rsday, 0.003] [u.s., 0.003] [billion, 0.003] [kraft, 0.003] [york, 0.003] [ago,
0.002] [city, 0.002] [kravis, 0.002] [administration, 0.002] [news, 0.002] [call
ed, 0.002]

Topic 6:
[percent, 0.005] [president, 0.004] [people, 0.003] [black-owned, 0.003] [saudi,
0.003] [u.s., 0.002] [economic, 0.002] [businesses, 0.002] [country, 0.002] [ira
q, 0.002] [central, 0.002] [government, 0.002] [officials, 0.002] [north, 0.002]
[dukakis, 0.002] [black, 0.002] [business, 0.002] [school, 0.002] [prices, 0.00
2] [nation, 0.002]

Topic 7:
[percent, 0.010] [soviet, 0.004] [economic, 0.003] [people, 0.003] [bush, 0.003]
[gorbachev, 0.003] [bank, 0.003] [fire, 0.003] [economy, 0.003] [national, 0.00
3] [u.s., 0.003] [tuesday, 0.003] [president, 0.003] [government, 0.003] [repor
t, 0.002] [monday, 0.002] [dukakis, 0.002] [israel, 0.002] [month, 0.002] [stoc
k, 0.002]

Topic 8:
[u.s., 0.004] [barry, 0.004] [officials, 0.004] [people, 0.003] [soviet, 0.003]
[government, 0.003] [city, 0.003] [friday, 0.003] [president, 0.003] [police, 0.
003] [moore, 0.003] [national, 0.002] [wednesday, 0.002] [central, 0.002] [thurs
day, 0.002] [north, 0.002] [visit, 0.002] [fbi, 0.002] [mayor, 0.002] [southern,
0.002]

Topic 9:
[percent, 0.015] [u.s., 0.004] [rate, 0.003] [noriega, 0.003] [rose, 0.003] [peo
ple, 0.003] [production, 0.003] [ago, 0.002] [october, 0.002] [november, 0.002]
[president, 0.002] [officials, 0.002] [bank, 0.002] [manufacturing, 0.002] [mont
h, 0.002] [oil, 0.002] [jackson, 0.002] [operating, 0.002] [global, 0.002] [gove
rnment, 0.002]

**Your Answers**

**Question 1:**
The differences between LDA and pLSA is that LDA is a fully generative probabilistic model, while pLSA is not. One cannot easily use pLSA to assign the probability of a class to a new unseen document, but one can in LDA. This is because LDA is modeled on a hidden Dirichlet variable that serves as a distribution of topics to documents, while pLSA is not. As a result of these changes, one advantage of LDA over pSLA is that LDA can better predict topics for new documents and suffers less from the overfitting issues that pSLA exhibits. However, it still retains the issue of needing to pick K before running the model.

**Question 2:**
For the demo, see code above. From the results, which were run for 10 topics, I observed some interesting results. Terms such as "people", "u.s.," and "president" appeared in nearly all of the topics as the words most likely to appear for that topic. This could be because these words appear often in the set of documents as a whole. However, looking past these frequently repeated words, we see that more distinct topics start to emerge: government, democracy, weather, international relations, etc.

LDA can be used for to calculate the likelihood that a document belongs to a particular topic, as well as analyze the mixture of topics present in a document through its different words. It can also be used in biology to analyze alleles that might be commonly carried by a group of individuals, or in recommendation systems to recommend a article to a customer based on his/her previously read articles (they form a "cluster" of sorts).

# End of Homework 6 :)

Please printout the Jupyter notebook and relevant code files that you work on and submit only 1 PDF file on GradeScope with page assigned.

nbm.py

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

class NB_model():
    def __init__(self):
        self.pi = {} # to store prior probability of each class
        self.Pr_dict = None
        self.num_vocab = None
        self.num_classes = None

    def fit(self, train_data, train_label, vocab, if_use_smooth=True):
        # get prior probabilities
        self.num_vocab = len(vocab['index'].tolist())
        self.get_prior_prob(train_label)
        # =================== YOUR CODE HERE ==========================
        # Calculate probability of each word based on class
        # Hint: Store each probability value in matrix or dict: self.Pr_dict[classID][wordID] or Pr_dict[wordID][classID])
        # Remember that there are possible NaN or 0 in Pr_dict matrix/dict. Use smooth method
        self.Pr_dict = {}
        self.zero_word_dict = {}
        classes = set(train_data['classIdx'])
        print(f"Training class ", end="", flush=True)
        for c in classes:
            df_class = train_data.loc[train_data['classIdx'] == c]
            print(f"{c} ", end="", flush=True)
            total = df_class['count'].sum()
            self.Pr_dict[c] = df_class.groupby('wordIdx')['count'].sum().to_dict()
            self.zero_word_dict[c] = 1 / (total + self.num_vocab)
            for k in self.Pr_dict[c]:
                self.Pr_dict[c][k] = (self.Pr_dict[c][k] + 1) / (total + self.num_vocab)
        # =============================================================
        print("\nTraining completed!")

    def predict(self, test_data):
        test_dict = test_data.to_dict() # change dataframe to dict
        new_dict = {}
        prediction = []

        for idx in range(len(test_dict['docIdx'])):
            docIdx = test_dict['docIdx'][idx]
            wordIdx = test_dict['wordIdx'][idx]
            count = test_dict['count'][idx]
            try:
                new_dict[docIdx][wordIdx] = count
            except:
                new_dict[test_dict['docIdx'][idx]] = {}
                new_dict[docIdx][wordIdx] = count
                ''
        for docIdx in range(1, len(new_dict)+1):
            if docIdx % 2500 == 0:
                print(f"Finished predictions for doc {docIdx}")
            score_dict = {}
            #Creating a probability row for each class
            for classIdx in range(1,self.num_classes+1):
                score_dict[classIdx] = 0
                # =================== YOUR CODE HERE ==========================
                ### Implement the score_dict for all classes for each document
                ### Remember to use log addtion rather than probability multiplication
                ### Remember to add prior probability, i.e. self.pi
                score_dict[classIdx] = np.log(self.pi[classIdx])
                for w in new_dict[docIdx]:
                    score_dict[classIdx] += np.log(self.Pr_dict[classIdx].get(w, self.zero_word_dict[classIdx]))
                # =============================================================
```

```
            max_score = max(score_dict, key=score_dict.get)
            prediction.append(max_score)
        return prediction


    def get_prior_prob(self,train_label, verbose=True):
        # print(train_label)
        unique_class = list(set(train_label))
        self.num_classes = len(unique_class)
        total = len(train_label)
        for c in unique_class:
            # =================== YOUR CODE HERE ===========================
            ### calculate prior probability of each class ####
            ### Hint: store prior probability of each class in self.pi
            self.pi[c] = 0
            for label in train_label:
                if c == label:
                    self.pi[c] += 1
            self.pi[c] = self.pi[c] / total
            # ==============================================================
        if verbose:
            print("Prior Probability of each class:")
            print("\n".join("{}: {}".format(k, v) for k, v in self.pi.items()))
```

document4.csv

```
,docIdx,wordIdx,count,words
395,4,10,1,atheist
396,4,12,20,of
397,4,16,1,from
398,4,17,10,religion
399,4,23,25,and
400,4,25,1,other
401,4,27,3,are
402,4,29,40,the
403,4,30,16,in
404,4,33,19,to
405,4,42,12,it
406,4,44,3,like
407,4,46,1,christians
408,4,48,2,on
409,4,51,6,but
410,4,52,7,with
411,4,54,2,word
412,4,60,22,is
413,4,67,5,people
414,4,72,3,can
415,4,73,1,get
416,4,81,4,for
417,4,83,6,who
418,4,84,4,go
419,4,85,1,directly
420,4,95,1,bible
421,4,99,2,so
422,4,100,3,one
423,4,101,1,such
424,4,104,3,by
425,4,122,4,or
426,4,139,1,an
427,4,142,4,which
428,4,143,1,may
429,4,144,13,be
430,4,152,1,humanism
431,4,156,3,secular
432,4,160,2,they
433,4,181,1,humanist
434,4,233,12,that
435,4,234,1,exists
436,4,235,4,all
437,4,239,3,any
438,4,245,1,well
439,4,251,6,this
440,4,277,1,example
441,4,279,1,anyone
442,4,282,1,use
443,4,289,2,many
444,4,291,2,thought
445,4,295,1,his
446,4,297,7,at
447,4,299,1,very
448,4,301,4,he
449,4,306,1,rather
450,4,307,1,than
451,4,309,2,although
452,4,310,1,often
453,4,312,1,had
454,4,314,4,some
455,4,316,2,god
456,4,340,1,when
457,4,343,9,faith
458,4,364,3,christianity
```

```
459,4,388,6,as
460,4,393,6,system
461,4,394,1,unfortunately
462,4,400,1,whose
463,4,410,1,premise
464,4,416,1,take
465,4,420,2,again
466,4,425,1,under
467,4,426,9,christian
468,4,438,2,only
469,4,449,1,world
470,4,451,1,down
471,4,458,1,more
472,4,465,1,work
473,4,466,3,has
474,4,469,1,however
475,4,470,2,probably
476,4,473,5,if
477,4,474,4,you
478,4,477,4,what
479,4,479,3,different
480,4,482,1,sure
481,4,489,1,christ
482,4,491,1,seems
483,4,492,2,even
484,4,515,1,university
485,4,529,1,belief
486,4,531,2,also
487,4,536,1,most
488,4,550,1,case
489,4,551,2,against
490,4,555,1,best
491,4,563,3,without
492,4,574,2,way
493,4,575,1,whether
494,4,588,1,emphasis
495,4,604,1,dictionary
496,4,612,1,present
497,4,613,4,person
498,4,614,1,philosophy
499,4,621,1,expressed
500,4,623,1,over
501,4,630,2,think
502,4,644,1,was
503,4,646,2,reason
504,4,657,1,western
505,4,658,1,values
506,4,663,2,were
507,4,683,1,through
508,4,692,1,those
509,4,695,2,beyond
510,4,722,9,not
511,4,731,1,rationalism
512,4,742,1,mind
513,4,746,1,become
514,4,748,1,them
515,4,749,6,there
516,4,766,3,will
517,4,770,1,article
518,4,775,4,edu
519,4,778,1,writes
520,4,779,2,science
521,4,793,4,humans
522,4,813,5,we
523,4,824,3,things
524,4,825,2,put
```

```
525,4,827,1,interesting
526,4,828,3,just
527,4,832,1,light
528,4,834,1,much
529,4,838,1,something
530,4,845,2,simply
531,4,847,1,understanding
532,4,849,2,ok
533,4,850,1,me
534,4,863,3,don
535,4,864,1,anyway
536,4,870,3,cs
537,4,877,2,do
538,4,886,2,no
539,4,887,1,didn
540,4,902,4,good
541,4,905,1,here
542,4,910,2,admit
543,4,912,2,up
544,4,914,1,definition
545,4,920,1,yes
546,4,921,1,now
547,4,922,6,have
548,4,930,2,out
549,4,939,1,within
550,4,942,2,should
551,4,947,3,define
552,4,965,1,fail
553,4,968,2,claimed
554,4,969,1,your
555,4,970,1,claim
556,4,977,1,every
557,4,982,1,implementation
558,4,985,1,seem
559,4,987,1,difference
560,4,988,1,between
561,4,992,2,still
562,4,993,1,say
563,4,995,1,my
564,4,996,2,personal
565,4,1003,5,would
566,4,1010,2,reasoning
567,4,1017,1,same
568,4,1018,6,because
569,4,1034,1,becomes
570,4,1038,2,point
571,4,1040,1,least
572,4,1044,1,notion
573,4,1065,3,someone
574,4,1071,2,course
575,4,1077,1,too
576,4,1101,1,might
577,4,1102,1,interpretation
578,4,1103,1,wouldn
579,4,1123,1,later
580,4,1124,1,going
581,4,1128,1,result
582,4,1130,1,yet
583,4,1134,1,kill
584,4,1137,1,everyone
585,4,1138,1,cause
586,4,1145,2,gun
587,4,1157,3,always
588,4,1160,2,basically
589,4,1196,3,doesn
590,4,1206,1,fair
```

```
591,4,1211,1,perhaps
592,4,1236,1,kills
593,4,1245,3,really
594,4,1260,3,prison
595,4,1266,1,irrelevant
596,4,1268,1,due
597,4,1270,1,concern
598,4,1282,1,inability
599,4,1283,1,totally
600,4,1305,1,similarly
601,4,1325,1,causes
602,4,1330,1,fun
603,4,1347,1,presumably
604,4,1365,1,free
605,4,1425,3,thinking
606,4,1452,3,cannot
607,4,1463,1,important
608,4,1472,1,realize
609,4,1485,1,responsibility
610,4,1487,1,maybe
611,4,1491,1,willing
612,4,1507,1,start
613,4,1508,1,ask
614,4,1552,1,bad
615,4,1557,1,problem
616,4,1571,1,let
617,4,1580,1,past
618,4,1608,1,real
619,4,1635,1,open
620,4,1646,4,themselves
621,4,1649,1,merely
622,4,1676,1,consider
623,4,1678,1,basic
624,4,1690,1,nature
625,4,1735,1,come
626,4,1784,1,condemn
627,4,1810,1,harm
628,4,1855,1,language
629,4,1861,1,room
630,4,1863,1,need
631,4,1870,1,whatever
632,4,1894,1,please
633,4,1898,1,universe
634,4,1906,2,change
635,4,1910,1,further
636,4,1924,1,jesus
637,4,2030,1,level
638,4,2038,1,happened
639,4,2045,1,apr
640,4,2095,1,event
641,4,2111,1,experience
642,4,2113,2,individual
643,4,2182,1,religions
644,4,2220,1,believes
645,4,2232,4,beliefs
646,4,2254,1,whereas
647,4,2444,2,unless
648,4,2465,1,words
649,4,2534,1,leave
650,4,2585,1,neat
651,4,2608,1,webster
652,4,2633,1,bias
653,4,2717,1,himself
654,4,2791,3,moment
655,4,2807,1,test
656,4,2829,2,game
```

```
657,4,2906,1,dedicated
658,4,3023,2,mass
659,4,3057,1,careful
660,4,3069,1,minded
661,4,3140,7,dogma
662,4,3153,1,inherently
663,4,3167,1,adequately
664,4,3183,3,philosopher
665,4,3192,1,sets
666,4,3289,1,adam
667,4,3290,1,john
668,4,3291,1,cooper
669,4,3292,1,verily
670,4,3293,1,laughed
671,4,3294,1,weaklings
672,4,3295,1,acooper
673,4,3296,1,macalstr
674,4,3297,1,claws
675,4,3469,4,prisoner
676,4,3471,3,genocide
677,4,3482,1,tradition
678,4,3501,1,dangerous
679,4,3559,1,correspond
680,4,3565,1,extend
681,4,3708,2,billions
682,4,3903,1,computer
683,4,3950,1,nice
684,4,3995,1,oriented
685,4,4066,1,regards
686,4,4088,2,ahead
687,4,4094,1,department
688,4,4316,1,divisions
689,4,4501,1,semitic
690,4,4709,1,rationality
691,4,4729,1,qualify
692,4,4743,3,understandable
693,4,4921,1,qualities
694,4,5003,1,leaves
695,4,5082,1,capable
696,4,5149,1,anybody
697,4,5488,1,granted
698,4,5915,1,sadly
699,4,6099,1,intuition
700,4,6427,1,suicide
701,4,6521,1,amazing
702,4,6625,1,bet
703,4,6954,1,guarding
704,4,7077,1,destroying
705,4,7670,4,encourages
706,4,7803,1,edt
707,4,7808,1,benevolence
708,4,7951,1,waco
709,4,8012,1,scorn
710,4,8142,1,seemingly
711,4,8413,1,bold
712,4,8811,1,sects
713,4,8979,1,moralities
714,4,9007,1,operates
715,4,9413,1,evaluated
716,4,9414,1,difficulty
717,4,9445,1,appalling
718,4,10050,1,visible
719,4,10310,1,colored
720,4,10528,1,testing
721,4,10775,4,toronto
722,4,11227,1,losing
```

```
723,4,11256,1,tests
724,4,11761,1,visiting
725,4,12034,1,offers
726,4,13012,1,framework
727,4,14705,1,skin
728,4,15869,1,constrained
729,4,17147,1,evaluate
730,4,18175,1,poking
731,4,19354,1,invested
732,4,20593,3,todd
733,4,22058,1,chest
734,4,22098,2,kelley
735,4,22508,1,retaining
736,4,31724,1,guise
737,4,31760,1,boil
738,4,31927,1,therein
739,4,36512,1,bullets
740,4,38811,1,nuances
741,4,41950,1,hypotheses
742,4,45754,1,debated
743,4,45956,1,pantheism
744,4,46815,1,arisen
745,4,53416,1,differentiated
746,4,53976,2,tgk
747,4,53977,1,quelled
748,4,53978,1,quintessential
749,4,53979,1,philanthropy
750,4,53980,3,supernaturalists
```

plsa.py

```
from numpy import zeros, int8, log
import numpy as np
from pylab import random
import sys
#import jieba
import nltk
from nltk.tokenize import word_tokenize
import re
import time
import codecs

class PLSA(object):
    def initialize(self, N, K, M, word2id, id2word, X):
        self.word2id, self.id2word, self.X = word2id, id2word, X
        self.N, self.K, self.M = N, K, M
        # theta[i, j] : p(zj|di): 2-D matrix
        self.theta = random([N, K])
        # beta[i, j] : p(wj|zi): 2-D matrix
        self.beta = random([K, M])
        # p[i, j, k] : p(zk|di,wj): 3-D tensor
        self.p = zeros([N, M, K])
        for i in range(0, N):
            normalization = sum(self.theta[i, :])
            for j in range(0, K):
                self.theta[i, j] /= normalization

        for i in range(0, K):
            normalization = sum(self.beta[i, :])
            for j in range(0, M):
                self.beta[i, j] /= normalization


    def EStep(self):
        for i in range(0, self.N): # w
            for j in range(0, self.M): # d
                ## ================== YOUR CODE HERE ==========================
                ###  for each word in each document, calculate its
                ###  conditional probability belonging to each topic (update p)
                # total = 0
                # for k in range(0, self.K):
                #     total += self.theta[i, k] * self.beta[k, j]

                # for k in range(self.K):
                #     self.p[i, j, k] = self.theta[i, k] * self.beta[k, j] / total

                # vectorized version
                total = np.matmul(self.theta[i], self.beta[:,j].T)
                self.p[i, j, :] = self.theta[i] * self.beta[:, j] / total
                # ============================================================

    def MStep(self):
        # update beta
        for k in range(0, self.K):
            # ================== YOUR CODE HERE ==========================
            ###  Implement M step 1: given the conditional distribution
            ###  find the parameters that can maximize the expected likelihood (update beta)
            # denominator = 0
            # for j in range(self.M):
            #     # for i in range(self.N):
            #     #     denominator += self.p[i, j, k] * self.X[i, j]
            #     denominator += np.matmul(self.p[:, j, k], self.X[:, j])

            # for j in range(self.M):
            #     # numerator = 0
```

```python
            #    # for i in range(self.N):
            #    #     numerator += self.p[i, j, k] * self.X[i, j]
            #    print(self.p[:,j,k].shape)
            #    print(self.X[:,j].shape)
            #    numerator = np.matmul(self.p[:, j, k], self.X[:, j])
            #    self.beta[k, j] = numerator / denominator

            # vectorized version
            denominator = np.einsum('ij,ij', self.p[:, :, k], self.X)
            self.beta[k] = np.einsum('i...,i...', self.p[:, :, k], self.X) / denominator
            # ============================================================

        # update theta
        for i in range(0, self.N):
            # ================== YOUR CODE HERE ==========================
            ###  Implement M step 2: given the conditional distribution
            ###  find the parameters that can maximize the expected likelihood (update theta)
            # for k in range(self.K):
            #    # numerator = 0
            #    # for j in range(self.M):
            #    #     numerator += self.p[i, j, k] * self.X[i, j]
            #    numerator = np.matmul(self.p[i, :, k], self.X[i, :].T)
            #    self.theta[i, k] = numerator / self.N

            # denominator = 0
            # for k in range(0, self.K):
            #    for j in range(0, self.M):
            #        denominator += self.p[i, j, k] * self.X[i, j]

            # vectorized version
            denominator = self.X[i,:].sum()
            numerator = np.matmul(self.p[i].T, self.X[i])
            self.theta[i] = numerator / denominator
            # ============================================================

    # calculate the log likelihood
    def LogLikelihood(self):
        loglikelihood = 0
        # for i in range(0, self.N):
        #    # for j in range(0, self.M):
        #        # # ================== YOUR CODE HERE ==========================
        #        # ###  Calculate likelihood function
        #        # # word_prob = 0
        #        # # for k in range(0, self.K):
        #        # #     word_prob += self.theta[i, k] * self.beta[k, j]
        #        # word_prob = np.matmul(self.theta[i], self.beta[:,j].T)
        #        # loglikelihood += self.X[i, j] * np.log(word_prob)
        #        # # ============================================================
        #    product = np.matmul(self.beta.T, self.theta[i])
        #    # loglikelihood += np.multiply(self.X[i], np.log(product)).sum()
        #    loglikelihood += np.einsum('i,i', self.X[i], np.log(product))

        # vectorized version
        product = np.matmul(self.theta, self.beta)
        loglikelihood += np.einsum('ij,ij', self.X, np.log(product))
        return loglikelihood

    # output the params of model and top words of topics to files
    def output(self, docTopicDist, topicWordDist, dictionary, topicWords, topicWordsNum):
        # document-topic distribution
        file = codecs.open(docTopicDist,'w','utf-8')
        for i in range(0, self.N):
            tmp = ''
            for j in range(0, self.K):
                tmp += str(self.theta[i, j]) + ' '
            file.write(tmp + '\n')
```

```
        file.close()

        # topic-word distribution
        file = codecs.open(topicWordDist,'w','utf-8')
        for i in range(0, self.K):
            tmp = ''
            for j in range(0, self.M):
                tmp += str(self.beta[i, j]) + ' '
            file.write(tmp + '\n')
        file.close()

        # dictionary
        file = codecs.open(dictionary,'w','utf-8')
        for i in range(0, self.M):
            file.write(self.id2word[i] + '\n')
        file.close()

        # top words of each topic
        file = codecs.open(topicWords,'w','utf-8')
        for i in range(0, self.K):
            topicword = []
            ids = self.beta[i, :].argsort()
            for j in ids:
                topicword.insert(0, self.id2word[j])
            tmp = ''
            for word in topicword[0:min(topicWordsNum, len(topicword))]:
                tmp += word + ' '
            file.write(tmp + '\n')
        file.close()
```