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: Ali Mirabzadeh, UID: 305179067

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 (https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html)](https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html).

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
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/alimirabzadeh/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" (http://qwone.com/~jason/20Newsgroups/)](http://qwone.com/~jason/20Newsgroups/). 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
test_data = open('./nb/dataset/test.data')
df_test = pd.read_csv(test_data, delimiter=' ', names=['docIdx', 'wordIdx',

# 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.

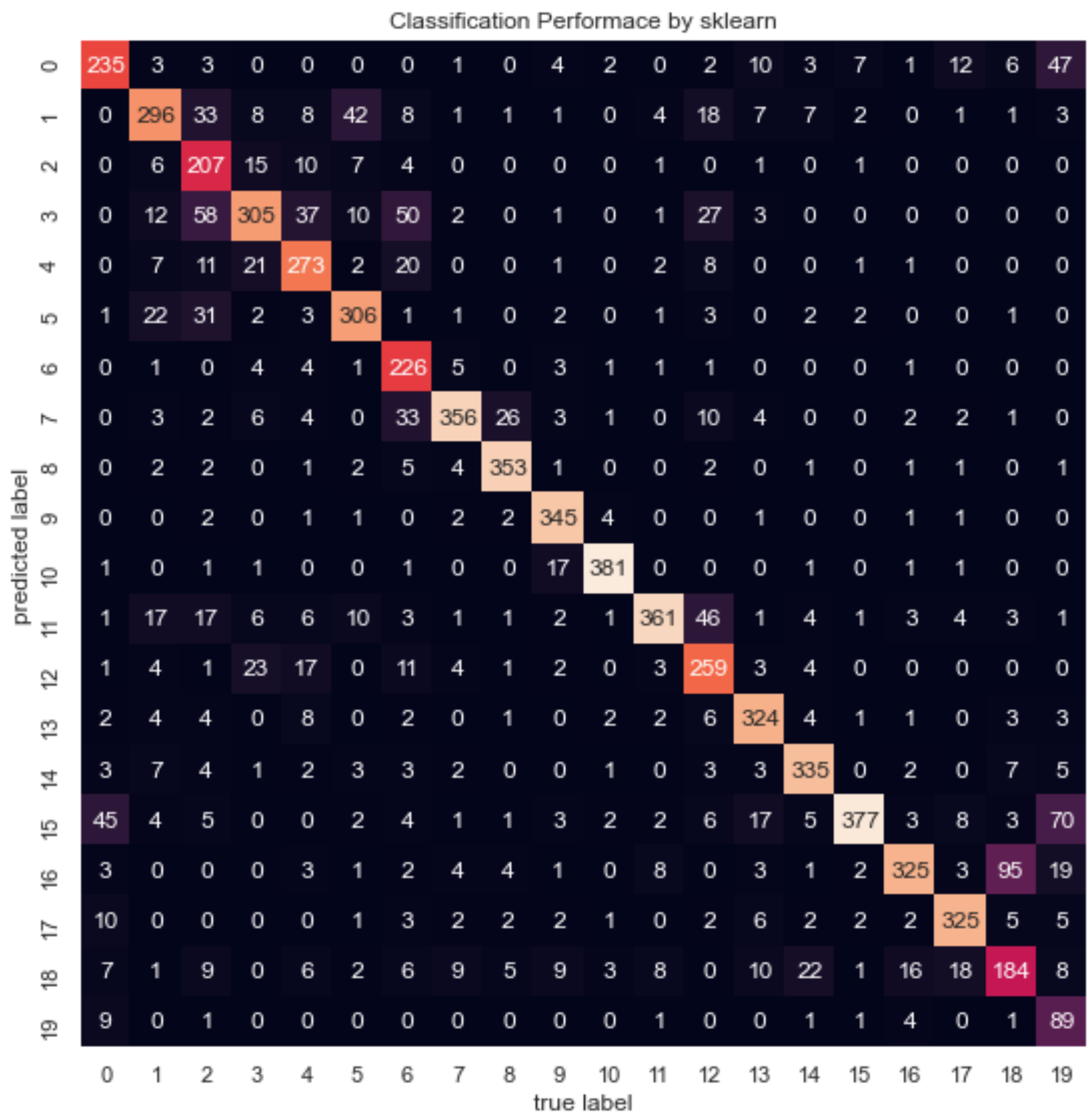
In [4]:
```
from nb.nbm import NB_model

# 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 completed!
```

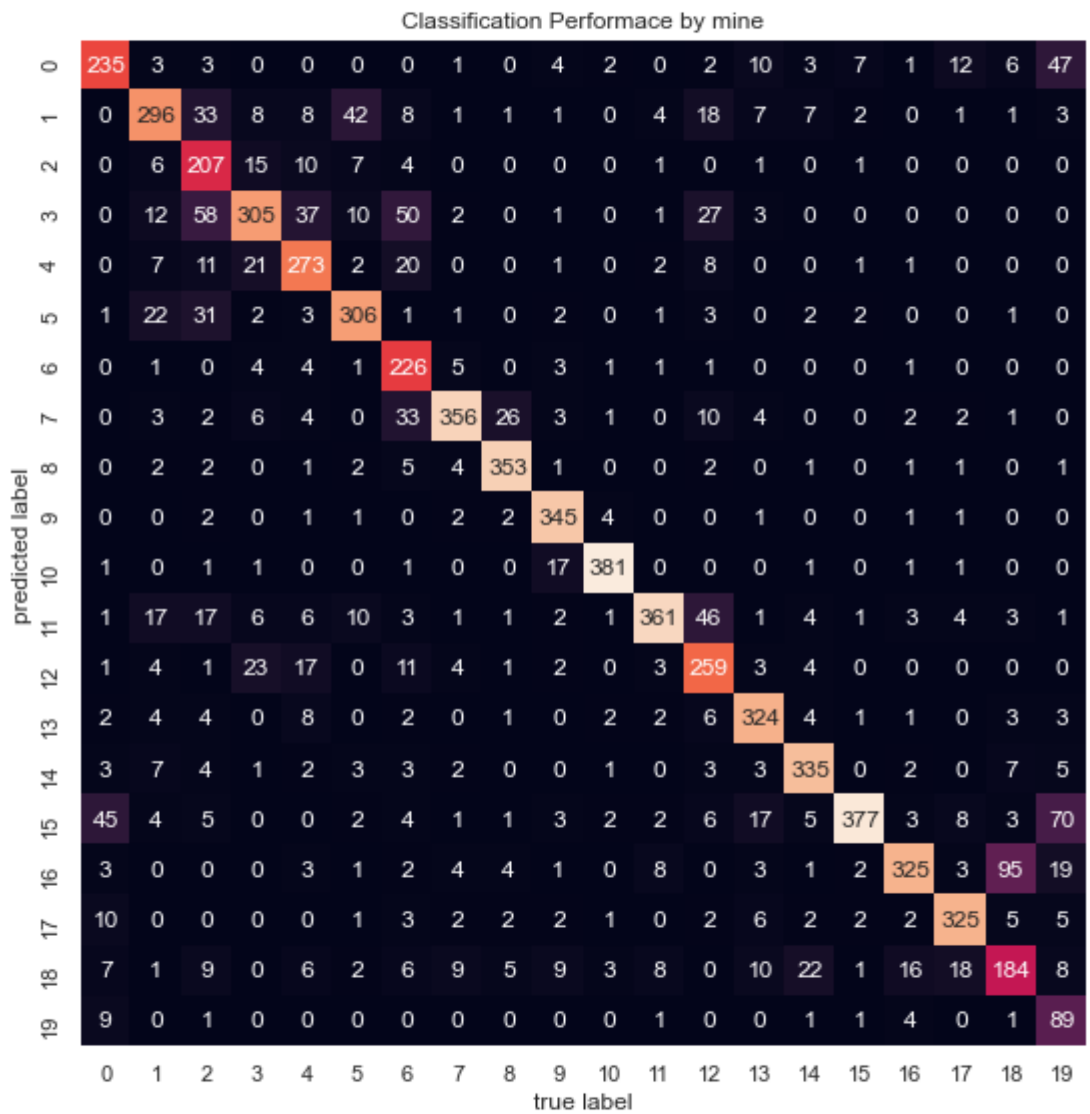
In [5]:
```
# make predictions on train set to validate the model
predict_train_labels = nbm.predict(df_train)
train_acc = (np.array(train_label) == np.array(predict_train_labels)).mean(
print("Accuracy on training data by my implementation: {}".format(train_acc

# 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)
```

```
Accuracy on training data by my implementation: 0.941077291685154
Accuracy on training data by my implementation: 0.7810792804796802
```

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 sklearn')
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.tight_layout()
plt.savefig('./nb/output/nbm_sklearn.png')
plt.show()
```



In [7]:
```
# 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 mine')
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.tight_layout()
plt.savefig('./nb/output/nbm_mine.png')
plt.show()
##They seem to be identical!
```



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/` .

```
ions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
Accuracy on training data by sklearn: 0.9326498143892522
Accuracy on test data by sklearn: 0.7738980350504514
```

Question & Analysis

0. Please indicate whether you implemented based the given code or from scratch.
1. 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	0.9326	0.7738
your implementaion	0.9411	0.7811

2. 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**
sit (2), couple (1), ...	Class 20	Class 3

3. 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?
4. 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).

Your Answers

0. Given Code
1. My implementation's results are pretty close to the one from Sklearn as we can in the above example. I think the reason for that is "Note that the dataset is slightly different loaded in nbm_sklearn.py". Also, the confusion matrixs seem identical
2. I basically used train.data and picked docID = 11269 and picked a few word such as sit and couple and looked for their corresponsing docID in test.data. Then I mapped the docI to the corresponding label using train.label and test.label and noticed for those word they were predicted class 20 even though they are class 3
3. It's a generative model becasue it learns from joint probablity distribution. NB assumes that each feature is conditionally independent where as logistic regression doesn't make the same assumption and in fact it uses conditional probablities.
PROS: It's relatively simple to implement and is useful for applications like identifying spas/non-spams emails. CONS: However, since as mentioned, the model makes assumption that the features are independent so could misclassify as well that's why it's called naive.

4. Yes, in fact it's really great for indentifying spam from normal emails/message. Like we can train a model that has both spam and non-spam emails, so the model can learn what words appear in spams find the probablities of each word

---

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 (https://en.wikipedia.org/wiki/Stop_word)](https://en.wikipedia.org/wiki/Stop_word) 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 [20]:
```
# input file, outpot files and parameters
datasetFilePath = './plsa/dataset/dataset2.txt' # or set as './plsa/dataset
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 = 10  # number of topic
maxIteration = 20 # maxIteration and threshold control the train process
threshold = 3
topicWordsNum = 10 # parameter for output
```

In [21]:
```
from plsa.plsa import PLSA
from plsa.utils import preprocessing

N, M, word2id, id2word, X = preprocessing(datasetFilePath, stopwordsFilePat
```

In [22]:
```
plsa_model = PLSA()
plsa_model.initialize(N, K, M, word2id, id2word, X)

oldLoglikelihood = 1
newLoglikelihood = 1
print ("K: ", K)
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(newLoglikelihood - oldLoglikelihood < threshold):
        #break
    oldLoglikelihood = newLoglikelihood

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

```
K:  10
[ 2020-12-12 20:45:48 ] 1 iteration -152813.14222663164
[ 2020-12-12 20:46:38 ] 2 iteration -150702.2084671577
[ 2020-12-12 20:47:27 ] 3 iteration -147769.08482924715
[ 2020-12-12 20:48:15 ] 4 iteration -144286.93004346848
[ 2020-12-12 20:49:03 ] 5 iteration -140920.01085965018
[ 2020-12-12 20:49:51 ] 6 iteration -138065.39935262286
[ 2020-12-12 20:50:39 ] 7 iteration -135761.93322620235
[ 2020-12-12 20:51:28 ] 8 iteration -133966.65665138207
[ 2020-12-12 20:52:17 ] 9 iteration -132604.4753048485
[ 2020-12-12 20:53:04 ] 10 iteration -131577.26133772268
[ 2020-12-12 20:53:52 ] 11 iteration -130798.00244629064
[ 2020-12-12 20:54:41 ] 12 iteration -130206.31420406947
[ 2020-12-12 20:55:30 ] 13 iteration -129751.51912562801
[ 2020-12-12 20:56:18 ] 14 iteration -129396.17958538682
[ 2020-12-12 20:57:07 ] 15 iteration -129115.51666607056
[ 2020-12-12 20:57:55 ] 16 iteration -128892.35183639737
[ 2020-12-12 20:58:44 ] 17 iteration -128713.02264192027
[ 2020-12-12 20:59:33 ] 18 iteration -128565.99063177603
[ 2020-12-12 21:00:21 ] 19 iteration -128461.21019689328
[ 2020-12-12 21:01:12 ] 20 iteration -128391.02602565885
```

In [8]:
```
plsa_model.output(docTopicDist, topicWordDist, dictionary, topicWords, topi
```

K trials for dataset1:

```
K:  2
[ 2020-12-12 19:52:11 ] 1 iteration -7919.6262395904005
[ 2020-12-12 19:52:12 ] 2 iteration -7849.457857168658
[ 2020-12-12 19:52:12 ] 3 iteration -7748.688566278395
[ 2020-12-12 19:52:12 ] 4 iteration -7649.049330932226
[ 2020-12-12 19:52:12 ] 5 iteration -7570.229444533049
[ 2020-12-12 19:52:13 ] 6 iteration -7515.863292371991
[ 2020-12-12 19:52:13 ] 7 iteration -7486.754431830214
[ 2020-12-12 19:52:13 ] 8 iteration -7468.153382467457
[ 2020-12-12 19:52:13 ] 9 iteration -7453.440627442476
[ 2020-12-12 19:52:13 ] 10 iteration -7439.133104330273
[ 2020-12-12 19:52:14 ] 11 iteration -7423.579265547843
[ 2020-12-12 19:52:14 ] 12 iteration -7410.424259313294
[ 2020-12-12 19:52:14 ] 13 iteration -7403.122683031999
[ 2020-12-12 19:52:14 ] 14 iteration -7399.443413341776
[ 2020-12-12 19:52:14 ] 15 iteration -7397.904025162465
[ 2020-12-12 19:52:15 ] 16 iteration -7396.827965428653
[ 2020-12-12 19:52:15 ] 17 iteration -7395.167380502538
[ 2020-12-12 19:52:15 ] 18 iteration -7394.080746167426
[ 2020-12-12 19:52:15 ] 19 iteration -7393.566915161791
[ 2020-12-12 19:52:16 ] 20 iteration -7393.219834120998
```

```
K:  3
[ 2020-12-12 19:53:46 ] 1 iteration -7901.390411485891
[ 2020-12-12 19:53:46 ] 2 iteration -7800.116753824728
[ 2020-12-12 19:53:46 ] 3 iteration -7656.417216100979
[ 2020-12-12 19:53:46 ] 4 iteration -7482.062455212207
[ 2020-12-12 19:53:47 ] 5 iteration -7310.070139050328
[ 2020-12-12 19:53:47 ] 6 iteration -7180.512572669283
[ 2020-12-12 19:53:47 ] 7 iteration -7095.806817014209
[ 2020-12-12 19:53:48 ] 8 iteration -7046.064177397884
[ 2020-12-12 19:53:48 ] 9 iteration -7019.3430791795945
[ 2020-12-12 19:53:48 ] 10 iteration -6996.0751705925295
[ 2020-12-12 19:53:49 ] 11 iteration -6970.001732099258
[ 2020-12-12 19:53:49 ] 12 iteration -6944.556252137331
[ 2020-12-12 19:53:49 ] 13 iteration -6929.911450845423
[ 2020-12-12 19:53:49 ] 14 iteration -6924.797974358775
[ 2020-12-12 19:53:50 ] 15 iteration -6922.550293576286
[ 2020-12-12 19:53:50 ] 16 iteration -6921.153524156658
[ 2020-12-12 19:53:50 ] 17 iteration -6920.682085754938
[ 2020-12-12 19:53:51 ] 18 iteration -6920.508744484476
[ 2020-12-12 19:53:51 ] 19 iteration -6920.419385760949
[ 2020-12-12 19:53:51 ] 20 iteration -6920.367391100283
```

```
K:  4
[ 2020-12-12 19:28:13 ] 1 iteration -7741.817334532188
[ 2020-12-12 19:28:13 ] 2 iteration -7567.8917657387165
[ 2020-12-12 19:28:14 ] 3 iteration -7399.0793349558535
[ 2020-12-12 19:28:14 ] 4 iteration -7249.489181636675
[ 2020-12-12 19:28:14 ] 5 iteration -7119.101651314833
[ 2020-12-12 19:28:15 ] 6 iteration -7014.786765565352
[ 2020-12-12 19:28:15 ] 7 iteration -6925.490809474547
[ 2020-12-12 19:28:15 ] 8 iteration -6842.5491207282475
[ 2020-12-12 19:28:16 ] 9 iteration -6773.174608455707
[ 2020-12-12 19:28:16 ] 10 iteration -6728.555918522748
[ 2020-12-12 19:28:16 ] 11 iteration -6704.522138456633
[ 2020-12-12 19:28:17 ] 12 iteration -6691.289441504023
[ 2020-12-12 19:28:17 ] 13 iteration -6684.534950399586
[ 2020-12-12 19:28:18 ] 14 iteration -6680.0189138696
[ 2020-12-12 19:28:18 ] 15 iteration -6676.102057629343
[ 2020-12-12 19:28:18 ] 16 iteration -6671.675016285936
[ 2020-12-12 19:28:19 ] 17 iteration -6666.545915827314
[ 2020-12-12 19:28:19 ] 18 iteration -6662.238776656173
[ 2020-12-12 19:28:19 ] 19 iteration -6660.300068320686
[ 2020-12-12 19:28:20 ] 20 iteration -6659.607678141125
```

K trials for datasets2:

```
K:  2
[ 2020-12-12 19:55:54 ] 1 iteration -153741.30982181325
[ 2020-12-12 19:56:05 ] 2 iteration -152830.88672493157
[ 2020-12-12 19:56:16 ] 3 iteration -151977.3835246241
[ 2020-12-12 19:56:27 ] 4 iteration -151229.49484554326
[ 2020-12-12 19:56:38 ] 5 iteration -150643.34143026551
[ 2020-12-12 19:56:50 ] 6 iteration -150211.7626220788
[ 2020-12-12 19:57:01 ] 7 iteration -149901.37494689875
[ 2020-12-12 19:57:12 ] 8 iteration -149679.8382707446
[ 2020-12-12 19:57:23 ] 9 iteration -149518.03058236607
[ 2020-12-12 19:57:34 ] 10 iteration -149396.424813981
[ 2020-12-12 19:57:46 ] 11 iteration -149300.89956035835
[ 2020-12-12 19:57:57 ] 12 iteration -149220.96021128865
[ 2020-12-12 19:58:08 ] 13 iteration -149151.43672967708
[ 2020-12-12 19:58:19 ] 14 iteration -149088.02518471918
[ 2020-12-12 19:58:30 ] 15 iteration -149031.8590827375
[ 2020-12-12 19:58:42 ] 16 iteration -148985.96587131688
[ 2020-12-12 19:58:53 ] 17 iteration -148949.2021389379
[ 2020-12-12 19:59:04 ] 18 iteration -148920.43603557497
[ 2020-12-12 19:59:15 ] 19 iteration -148896.71806280757
[ 2020-12-12 19:59:26 ] 20 iteration -148874.20020718026
```

```
K:  3
[ 2020-12-12 20:10:31 ] 1 iteration -153507.93675527006
[ 2020-12-12 20:10:46 ] 2 iteration -152283.65440721923
[ 2020-12-12 20:11:01 ] 3 iteration -150904.24638959861
[ 2020-12-12 20:11:17 ] 4 iteration -149541.38862566577
[ 2020-12-12 20:11:33 ] 5 iteration -148365.17278460204
[ 2020-12-12 20:11:49 ] 6 iteration -147414.2755639266
[ 2020-12-12 20:12:04 ] 7 iteration -146666.59358327917
[ 2020-12-12 20:12:20 ] 8 iteration -146086.9223074089
[ 2020-12-12 20:12:36 ] 9 iteration -145646.452292398
[ 2020-12-12 20:12:51 ] 10 iteration -145323.4856398951
[ 2020-12-12 20:13:06 ] 11 iteration -145095.21431161382
[ 2020-12-12 20:13:22 ] 12 iteration -144933.12441075433
[ 2020-12-12 20:13:38 ] 13 iteration -144811.18897993598
[ 2020-12-12 20:13:54 ] 14 iteration -144712.65284691955
[ 2020-12-12 20:14:09 ] 15 iteration -144636.4382910918
[ 2020-12-12 20:14:25 ] 16 iteration -144580.80182003425
[ 2020-12-12 20:14:40 ] 17 iteration -144540.0237754329
[ 2020-12-12 20:14:56 ] 18 iteration -144509.36156574343
[ 2020-12-12 20:15:11 ] 19 iteration -144481.25919719363
[ 2020-12-12 20:15:26 ] 20 iteration -144454.04203414303
```

```
K:  4
[ 2020-12-12 20:23:51 ] 1 iteration -153150.16363130286
[ 2020-12-12 20:24:10 ] 2 iteration -151574.13582451956
[ 2020-12-12 20:24:30 ] 3 iteration -149662.70289579206
[ 2020-12-12 20:24:50 ] 4 iteration -147632.2332598618
[ 2020-12-12 20:25:10 ] 5 iteration -145848.33577260995
[ 2020-12-12 20:25:30 ] 6 iteration -144492.17210519378
[ 2020-12-12 20:25:50 ] 7 iteration -143533.16818947604
[ 2020-12-12 20:26:09 ] 8 iteration -142855.4735421343
[ 2020-12-12 20:26:29 ] 9 iteration -142355.54309563778
[ 2020-12-12 20:26:50 ] 10 iteration -141979.8650948237
[ 2020-12-12 20:27:11 ] 11 iteration -141699.89628784102
[ 2020-12-12 20:27:30 ] 12 iteration -141487.4629628496
[ 2020-12-12 20:27:51 ] 13 iteration -141322.9215173641
[ 2020-12-12 20:28:10 ] 14 iteration -141196.0950263997
[ 2020-12-12 20:28:31 ] 15 iteration -141094.0518121649
[ 2020-12-12 20:28:51 ] 16 iteration -141007.1258854785
[ 2020-12-12 20:29:11 ] 17 iteration -140934.09464703494
[ 2020-12-12 20:29:31 ] 18 iteration -140874.1458193671
[ 2020-12-12 20:29:52 ] 19 iteration -140827.51683027574
[ 2020-12-12 20:30:13 ] 20 iteration -140791.882043118
```

Question & Analysis

0. Please indicate whether you implemented based the given code or from scratch.
1. 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
luffy devil pirates fruit piece '' `` manga user fruits	luffy crew alabasta baroque navy ace d. pirates pirate roger	grand sea haki called island burū mountain pose blue red	luffy pirates crew island straw dressrosa franky alliance zou hats

For dataset 2:

Topic 1	Topic 2	Topic 3	Topic 4
`` '' percent soviet u.s. officials oil rate monday prices	`` '' bank percent soviet people gorbachev government billion economy	`` '' u.s. official government people president noriega roberts united	`` '' bush dukakis people campaign percent president company california

2. Are there any similarities between pLSA and GMM model? Briefly explain your thoughts.
3. What are the disadvantages of pLSA? Consider its generalizing ability to new unseen document and its parameter complexity, etc.

Your Answers

0. Given Code
1. Based on the Piazza post I tried different K's from {2,3,4} and checking the topics results, k=4 seem to be reasonable
2. Yes, they both use EM algorithm and probablity distribution. PLSA is applicable for text data.
3. I think tuning the hyperparameters is a disadvantage as for large datasets, like dataset2, here, it could take hours to find the best parameters. pLSA is not good at generalizing new data so this could results in not performing well on new unseen data

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) (https://ai.stanford.edu/~ang/papers/nips01-lda)](https://ai.stanford.edu/~ang/papers/nips01-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.

Your Answers Used: https://medium.com/nanonets/topic-modeling-with-lsa-psla-lda-and-

lda2vec-
555ff65b0b05#:~:text=LDA%20typically%20works%20better%20than,fixed%20point%20in%20the%
(https://medium.com/nanonets/topic-modeling-with-lsa-psla-lda-and-lda2vec-
555ff65b0b05#:~:text=LDA%20typically%20works%20better%20than,fixed%20point%20in%20the%

1. "LDA typically works better than pLSA because it can generalize to new documents easily." As mentioned plsa is not good at generalizing new unseen data so that's an advantage of LDA over pLSA

In [ ]:

```
import nltk
import gensim
```

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.

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

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.classes = collections.defaultdict(int)

        word_count_per_class = collections.defaultdict(lambda: collections.defaultdict(int))
        self.Pr_dict = collections.defaultdict(lambda: collections.defaultdict(float))

        train_dict = train_data.to_dict()
        for i in range(len(train_dict['classIdx'])):
            self.classes[train_dict['classIdx'][i]] += train_dict['count'][i]
            word_count_per_class[train_dict['classIdx'][i]][train_dict['wordIdx'][i]] += train_dict['count'][i]

        for classID in word_count_per_class:
            for wordID in word_count_per_class[classID]:
                self.Pr_dict[classID][wordID] = (word_count_per_class[classID][wordID] + 1) /
                                                                      (self.classes[classID] + self.num_vocab)
        # ============================================================
        print("Training 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):
```

```
            score_dict = {}
            max_score = 0
            #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 wordId in new_dict[docIdx]:
                    if self.Pr_dict[classIdx][wordIdx] == 0:
                        score_dict[classIdx] += new_dict[docIdx][wordId] * np.log(1/(self.classes[classIdx] + self.num_vocab))
                    else:
                        score_dict[classIdx] += new_dict[docIdx][wordId] * np.log(self.Pr_dict[classIdx][wordId])
                # ============================================================
            max_score = max(score_dict, key=score_dict.get)
            prediction.append(max_score)
        return prediction


    def get_prior_prob(self,train_label, verbose=True):
        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
            counter = 0
            for label in train_label:
                if c is label:
                    counter += 1
            self.pi[c] = counter / total
            # ============================================================
        if verbose:
            print("Prior Probability of each class:")
            print("\n".join("{}: {}".format(k, v) for k, v in self.pi.items()))
```

```
from numpy import zeros, int8, log
from pylab import random
import sys
#import jieba
import nltk
from nltk.tokenize import word_tokenize
import re
import time
import codecs
# N is # of of document
# K is # of topic
# M is # of word
# beta is probablity of word given a topic
# theta is probablity of a topic given a document
# document- word matrix, N x M : word count in a document
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):
            for j in range(0, self.M):
                ## ================== YOUR CODE HERE ==========================
                ###  for each word in each document, calculate its
                ###  conditional probability belonging to each topic (update p)
                denominator = 0
                for k in range(0, self.K):
                    self.p[i, j, k] = self.theta[i, k] * self.beta[k, j]
                    denominator += self.p[i, j, k]
                for k in range(0, self.K):
                    self.p[i, j, k] /= denominator
                # ============================================================

    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
```

```python
            denominator = 0
            for m in range(0, self.M):
                self.beta[k, m] = 0
                for n in range(0, self.N):
                    self.beta[k, m] += self.X[n, m] * self.p[n, m, k]
                denominator += self.beta[k, m]
            for m in range(0, self.M):
                self.beta[k, m] /= 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(0, self.K):
                self.theta[i, k] = 0
                denominator = 0
                for m in range(0, self.M):
                    self.theta[i, k] += self.X[i, m] * self.p[i, m, k]
                    denominator += self.X[i, m]
                self.theta[i, k] /= 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
                temp = 0
                for k in range(0, self.K):
                    temp += self.theta[i, k] * self.beta[k, j]
                if temp > 0:
                    loglikelihood += self.X[i, j] * log(second_term)
                # ============================================================
        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 = ''
```

```
                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()
```