HW 5: Clustering and Topic Modeling

Each assignment needs to be completed independently. Never ever copy others' work (even with minor modification, e.g. changing variable names). Anti-Plagiarism software will be used to check all submissions.

In this assignment, you'll practice different text clustering methods. A dataset has been prepared for you:

- `hw5_train.csv` : This file contains a list of documents. It's used for training models
- `hw5_test` : This file contains a list of documents and their ground-truth labels (4 lables: 1,2,3,7). It's used for external evaluation.

Text	Label
paraglider collides with hot air balloon ...	1
faa issues fire warning for lithium ...	2
....	...

Sample outputs have been provided to you. Due to randomness, you may not get the exact result as shown here, but your result should be close if you tune the parameters carefully. Your taget is to `achieve above 70% F1 on the test dataset`

Q1: K-Mean Clustering

Define a function `cluster_kmean(train_text, test_text, text_label)` as follows:

- Take three inputs:
  - `train_text` is a list of documents for traing
  - `test_text` is a list of documents for test
  - `test_label` is the labels corresponding to documents in `test_text`
- First generate `TFIDF` weights. You need to decide appropriate values for parameters such as `stopwords` and `min_df` :
  - Keep or remove stopwords? Customized stop words?
  - Set appropriate `min_df` to filter infrequent words
- Use `KMeans` to cluster documents in `train_text` into 4 clusters. Here you need to decide the following parameters:
  - Distance measure: `cosine similarity` or `Euclidean distance` ? Pick the one which gives you better performance.
  - When clustering, be sure to use sufficient iterations with different initial centroids to make sure clustering converge.
- Test the clustering model performance using `test_label` as follows:
  - Predict the cluster ID for each document in `test_text` .
  - Apply `majority vote` rule to dynamically map the predicted cluster IDs to `test_label` . Note, you'd better not hardcode the mapping, because cluster IDs may be assigned differently in each run. (hint: if you use pandas, look for `idxmax` function).
  - Print out the cross tabluation between cluster ids and class labels
  - print out the classification report for the test subset
- This function has no return. Print out the classification report.
- Briefly discuss:
  - Which distance measure is better and why it is better.
  - Could you assign a meaningful name to each cluster? Discuss how you interpret each cluster.
- Write your analysis in a pdf file.

In [1]:
```python
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn import metrics, mixture
from nltk.corpus import stopwords
from nltk.cluster import KMeansClusterer, cosine_distance
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from nltk.corpus import stopwords
from sklearn.decomposition import LatentDirichletAllocation

# Add your import statement
```

In [2]:
```python
train = pd.read_csv("hw5_train.csv")
train.head()

test = pd.read_csv("hw5_test.csv")
test.head()

test_text = test["text"]
test_label = test["label"]
test_text
```

Out[2]:
```
0       No desire to visit mother in jail, am I a bad ...
1       what types of desirable products/materials can...
2       what is teleportation? why an unknown indian i...
3       Do you have to read the whole Bible to get int...
4       6 yr old son with a Deviated Septum!!!!?\nMy s...
                              ...
1269    most ghetto.?\nI'm aware that ghetto is an ign...
1270    Does anyone know about free on-line Wiccan new...
1271    total number of cell divisions from the beginn...
1272    What should I do to relieve my coughing and di...
1273    what does this sound like to you.........?\nso...
Name: text, Length: 1274, dtype: object
```

In [18]:
```python
def cluster_kmean(train, test_text, test_label):

   # Add your code
    tfidf_vect = TfidfVectorizer(stop_words = "english", min_df = 5)
    dtm = tfidf_vect.fit_transform(train)
    print(dtm.shape)

    num_clusters = 4

    clusterer = KMeansClusterer(num_clusters, cosine_distance, repeats = 20)
    clusters = clusterer.cluster(dtm.toarray(), assign_clusters = True)
    test_dtm = tfidf_vect.transform(test_text)
    predicted = [clusterer.classify(v) for v in test_dtm.toarray()]

    confusion_df = pd.DataFrame(list(zip(test_label, predicted)), columns = ["label", "cluster"])
    crosstab = pd.crosstab(index = confusion_df.cluster, columns = confusion_df.label)
    print('Crosstab of Kmean \n',crosstab)

    clusters_dict = crosstab.idxmax(axis = 'columns')
    predicted_targets = [clusters_dict[i] for i in predicted]

    print('\n Classification Report of Kmean\n',metrics.classification_report(test_label, predicted_targets))
```

In [19]:
```python
%%time
cluster_kmean(train["text"], test_text, test_label)
```

```
(4000, 6861)
Crosstab of Kmean
 label      1    2    3    7
cluster
0          7  223   27    7
1         33   30   21  199
2         61   48  293   45
3        231   13   14   22

 Classification Report of Kmean
               precision    recall  f1-score   support

           1       0.82      0.70      0.75       332
           2       0.84      0.71      0.77       314
           3       0.66      0.83      0.73       355
           7       0.70      0.73      0.72       273

    accuracy                           0.74      1274
   macro avg       0.76      0.74      0.74      1274
weighted avg       0.76      0.74      0.74      1274

CPU times: total: 8min 56s
Wall time: 2min 18s
```

In [5]:
```python
# cluster_kmean(train["text"], test_text, test_label)
```

Q2: Clustering by Gaussian Mixture Model

In this task, you'll re-do the clustering using a Gaussian Mixture Model. Call this function `cluster_gmm(train_text, test_text, text_label)` .

Write your analysis on the following:

- How did you pick the parameters such as the number of clusters, variance type etc.?
- Compare to Kmeans in Q1, do you achieve better preformance by GMM?
- Note, like KMean, be sure to use different initial means (i.e. `n_init` parameter) when fitting the model to achieve the model stability

In [24]:
```python
def cluster_gmm(train, test_text, test_label):

    # Add your code
    tfidf_vect = TfidfVectorizer(stop_words = "english", min_df = 5)
    dtm = tfidf_vect.fit_transform(train)
    lowest_bic = np.infty
    best_gmm = None
    n_components_range = range(2,8)
    cv_types = ['spherical', 'tied', 'diag']

    for cv_type in cv_types:
        for n_component in n_components_range:
            gmm = mixture.GaussianMixture(n_components = n_component, covariance_type = cv_type, random_state = 42, n_init = 2)
            gmm.fit(dtm.toarray())
            bic = gmm.bic(dtm.toarray())
            if bic < lowest_bic:
                lowest_bic = bic
                best_gmm = gmm

    test_dtm = tfidf_vect.transform(test_text)
    predicted = best_gmm.predict(test_dtm.toarray())

    confusion_df = pd.DataFrame(list(zip(test_label, predicted)), columns = ["label", "cluster"])
    crosstab = pd.crosstab(index = confusion_df.cluster, columns = confusion_df.label)
    print('Crosstab \n',crosstab)

    clusters_dict = crosstab.idxmax(axis = 'columns')
    predicted_targets = [clusters_dict[i] for i in predicted]

    print('\nClassification Report for GMM\n',metrics.classification_report(test_label, predicted_targets))
```

In [25]:
```python
%%time
cluster_gmm(train["text"], test_text, test_label)
```

```
Crosstab
 label      1    2    3    7
cluster
0         67    1    1    1
1          0    0   27    2
2          0   20    0    0
3        160   37   97   77
4         10   16  173    9
5         11    3    5  116
6         84  237   52   68

Classification Report for GMM
               precision    recall  f1-score   support

           1       0.51      0.68      0.59       332
           2       0.56      0.82      0.66       314
           3       0.84      0.56      0.68       355
           7       0.86      0.42      0.57       273

    accuracy                           0.63      1274
   macro avg       0.69      0.62      0.62      1274
weighted avg       0.69      0.63      0.63      1274

CPU times: total: 1h 30min 2s
Wall time: 12min 53s
```

Q3: Clustering by LDA

In this task, you'll re-do the clustering using LDA. Call this function `cluster_lda(train_text, test_text, text_label)` .

However, since LDA returns topic mixture for each document, you `assign the topic with highest probability to each test document` , and then measure the performance as in Q1

In addition, within the function, please print out the top 30 words for each topic

Finally, please analyze the following:

- Based on the top words of each topic, could you assign a meaningful name to each topic?
- Although the test subset shows there are 4 clusters, without this information, how do you choose the number of topics?
- Does your LDA model achieve better performance than KMeans or GMM?

In [22]:
```python
def cluster_lda(train, test_text, test_label):

    # add your code
    tf_vectorizer = CountVectorizer(min_df = 5, stop_words=list(stopwords.words('english')))
    tf = tf_vectorizer.fit_transform(train.tolist()+test_text.tolist())
    tf_feature_names = tf_vectorizer.get_feature_names()
    num_topics = 4
    lda = LatentDirichletAllocation(n_components=num_topics, max_iter=30, verbose=1, evaluate_every=1, n_jobs=1, random_state=0).fit(tf)
    num_top_words = 30

    for topic_idx, topic in enumerate(lda.components_):
        print ("Topic %d:" % (topic_idx))
        words=[tf_feature_names[i] for i in topic.argsort()[::-1][0:num_top_words]]
        print(words)
        print("\n")

    topic_assign = lda.transform(tf)
    predicted = []

    for i in range(4000, 5274):
        x = np.array(topic_assign[i])
        predicted.append(np.argmax(x))

    confusion_df = pd.DataFrame(list(zip(test_label, predicted)), columns = ["label", "cluster"])
    crosstab = pd.crosstab(index = confusion_df.cluster, columns = confusion_df.label)
    print('Crosstab for LDA \n',crosstab)

    clusters_dict = crosstab.idxmax(axis = 'columns')
    predicted_targets = [clusters_dict[i] for i in predicted]

    print('\n Classification for LDA \n',metrics.classification_report(test_label, predicted_targets))
```

In [23]:
```python
%%time
cluster_lda(train["text"], test_text, test_label)
```

```
C:\Users\siddh\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will
be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
iteration: 1 of max_iter: 30, perplexity: 3429.2361
iteration: 2 of max_iter: 30, perplexity: 3228.8556
iteration: 3 of max_iter: 30, perplexity: 3051.9925
iteration: 4 of max_iter: 30, perplexity: 2921.8382
iteration: 5 of max_iter: 30, perplexity: 2836.1184
iteration: 6 of max_iter: 30, perplexity: 2778.0444
iteration: 7 of max_iter: 30, perplexity: 2735.2839
iteration: 8 of max_iter: 30, perplexity: 2704.0936
iteration: 9 of max_iter: 30, perplexity: 2682.0117
iteration: 10 of max_iter: 30, perplexity: 2665.9518
iteration: 11 of max_iter: 30, perplexity: 2653.2678
iteration: 12 of max_iter: 30, perplexity: 2642.9544
iteration: 13 of max_iter: 30, perplexity: 2634.0353
iteration: 14 of max_iter: 30, perplexity: 2626.5810
iteration: 15 of max_iter: 30, perplexity: 2620.8454
iteration: 16 of max_iter: 30, perplexity: 2616.2603
iteration: 17 of max_iter: 30, perplexity: 2612.0218
iteration: 18 of max_iter: 30, perplexity: 2608.3306
iteration: 19 of max_iter: 30, perplexity: 2605.2268
iteration: 20 of max_iter: 30, perplexity: 2602.5069
iteration: 21 of max_iter: 30, perplexity: 2600.1447
iteration: 22 of max_iter: 30, perplexity: 2598.1423
iteration: 23 of max_iter: 30, perplexity: 2596.3908
iteration: 24 of max_iter: 30, perplexity: 2594.8772
iteration: 25 of max_iter: 30, perplexity: 2593.5296
iteration: 26 of max_iter: 30, perplexity: 2592.2648
iteration: 27 of max_iter: 30, perplexity: 2591.0814
iteration: 28 of max_iter: 30, perplexity: 2589.8911
iteration: 29 of max_iter: 30, perplexity: 2588.6920
iteration: 30 of max_iter: 30, perplexity: 2587.5671
Topic 0:
['water', 'weight', 'eat', 'would', 'one', 'energy', '10', 'fat', 'nthe', 'air', 'light', 'number', 'earth', 'diet', 'time', 'like', 'two', 'also', 'lose', 'need', 'get', '000', 'd
ay', 'body', 'mass', 'go', 'help', 'know', 'speed', 'food']


Topic 1:
['also', 'body', 'may', 'doctor', 'blood', 'one', 'get', 'used', 'pain', 'like', 'skin', 'cause', 'help', 'could', 'cancer', 'use', 'take', 'see', 'disease', 'would', 'cells', 'nor
mal', 'many', 'brain', 'two', 'people', 'called', 'symptoms', 'first', 'well']


Topic 2:
['god', 'people', 'one', 'would', 'believe', 'us', 'jesus', 'think', 'question', 'world', 'like', 'know', 'life', 'nhttp', 'bible', 'many', 'religion', 'com', 'www', 'man', 'say',
'time', 'christians', 'see', 'nthe', 'even', 'word', 'name', 'way', 'church']


Topic 3:
['get', 'like', 'know', 'would', 'want', 'good', 'people', 'one', 'need', 'think', 'help', 'go', 'time', 'work', 'make', 'really', 'way', 'find', 'much', 'could', 'something', 'tak
e', 'money', 'feel', 'also', 'going', 'business', 'job', 'see', 'even']


Crosstab for LDA
 label      1    2    3    7
cluster
0          3  177   46    8
1          3   66  145    3
2        221   27    8    8
3        105   44  156  254

 Classification for LDA
               precision    recall  f1-score   support

           1       0.84      0.67      0.74       332
           2       0.76      0.56      0.65       314
           3       0.67      0.41      0.51       355
           7       0.45      0.93      0.61       273

    accuracy                           0.63      1274
   macro avg       0.68      0.64      0.63      1274
weighted avg       0.69      0.63      0.62      1274

CPU times: total: 1min 40s
Wall time: 1min 45s
```

In [10]:
```python
# cluster_lda(train["text"], test_text, test_label)
```

Q4 (Bonus): Topic Coherence and Separation

For the LDA model you obtained at Q3, can you measure the coherence and separation of topics? Suppose you have the following topics:

- Topic 1 keywords: business, money, company, pay, credit
- Topic 2 keywords: energy, earth, gas, heat, sun

Describe your ideas and implement them.

In [26]:
```python
if __name__ == "__main__":

    # Due to randomness, you won't get the exact result
    # as shown here, but your result should be close
    # if you tune the parameters carefully

    train = pd.read_csv("hw5_train.csv")
    train.head()

    test = pd.read_csv("hw5_test.csv")
    test.head()

    test_text = test["text"]
    test_label = test["label"]

    # Q1
    cluster_kmean(train["text"], test_text, test_label)

    # Q2
    cluster_gmm(train["text"], test_text, test_label)

    # Q2
    cluster_lda(train["text"], test_text, test_label)
```

In [ ]: