HW 5: Clustering and Topic Modeling

```
<div class="alert alert-block alert-warning">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. </div>
```

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]:

```
import pandas as pd
from nltk.corpus import stopwords
from sklearn import metrics
from sklearn.mixture import GaussianMixture
import numpy as np
from nltk.cluster import KMeansClusterer,cosine_distance
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer

# Add your import statement
```

In [2]:

```
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"]
```

In [3]:

```
train_data=pd.read_csv('C:/Users/Rossshhh/Downloads/hw5_train.csv')
train_data.head()
```

Out[3]:

	text
0	Would you rather get a gift that you knew what...
1	Is the internet ruining people's ability to co...
2	Permanganate?\nSuppose permanganate was used t...
3	If Rock-n-Roll is really the work of the devil...
4	Has anyone purchased software to watch TV on y...

In [4]:

```
test_data=pd.read_csv('C:/Users/Rossshhh/Downloads/hw5_test.csv')
test_data.head()
```

Out[4]:

	label	text
0	3	No desire to visit mother in jail, am I a bad ...
1	7	what types of desirable products/materials can...
2	2	what is teleportation? why an unknown indian i...
3	1	Do you have to read the whole Bible to get int...
4	3	6 yr old son with a Deviated Septum!!!!?\nMy s...

In [5]:

```
def cluster_kmean(train, test_text, test_label):

   # Add your code
    TFIDF_vect = TfidfVectorizer(stop_words="english",min_df=6)
    dtm= TFIDF_vect.fit_transform(train)
    num_clusters=4
    clust = KMeansClusterer(num_clusters,cosine_distance,repeats=10)
    cluster = clust.cluster(dtm.toarray(), assign_clusters=True)
    test_dtm = TFIDF_vect.transform(test_text)
    predicted = [clust.classify(v) for v in test_dtm.toarray()]
    df_confu = pd.DataFrame(list(zip(test_label.values, predicted)),columns = ["actual_class", "cluster"])
    final_res=pd.crosstab( index=df_confu.cluster, columns=df_confu.actual_class)
    print(final_res)
    final_res=final_res.reset_index()
    a=final_res.idxmax(axis=1)
    dict_cluster={}
    for data in range(len(final_res)):
        dict_cluster[final_res['cluster'][data]]=a[data]
    predicted_target=[dict_cluster[data] for data in predicted]
    print(metrics.classification_report(test_label, predicted_target))
```

In [6]:

```
cluster_kmean(train["text"], test_text, test_label)
```

```
actual_class    1    2    3    7
cluster
0               6  209    6    9
1              40   34   20  205
2             280   39   61   51
3               6   32  268    8
              precision    recall  f1-score   support

           1       0.65      0.84      0.73       332
           2       0.91      0.67      0.77       314
           3       0.85      0.75      0.80       355
           7       0.69      0.75      0.72       273

    accuracy                           0.76      1274
   macro avg       0.77      0.75      0.76      1274
weighted avg       0.78      0.76      0.76      1274
```

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 [7]:

```
TFIDF_vect = TfidfVectorizer(stop_words="english",min_df=5)
data_dtm= TFIDF_vect.fit_transform(train['text'])
```

In [8]:

```
bic_low = np.infty
gmm= None

n_compo_range = range(3,5)
cv_types = ['spherical', 'tied', 'diag']

for cv_t in cv_types:
    for n_components in n_compo_range:
        gmm_best = GaussianMixture(n_init=7,n_components=n_components,covariance_type=cv_t, random_state=42)
        gmm_best.fit(data_dtm.toarray())
        bic = gmm_best.bic(data_dtm.toarray())
        if bic < bic_low:
            bic_low = bic
            gmm = gmm_best
```

In [9]:

```
gmm
```

Out[9]:

```
GaussianMixture(covariance_type='diag', n_components=4, n_init=7,
                random_state=42)
```

In [10]:

```
dtm_testing = TFIDF_vect.transform(test["text"])
predictt =gmm.predict(dtm_testing.toarray())
predictt[0:5]
```

Out[10]:

```
array([1, 0, 2, 1, 1], dtype=int64)
```

In [11]:

```
confusion_df = pd.DataFrame(list(zip(test["label"].values, predictt)),columns = ["actual_class", "cluster"])
res=pd.crosstab( index=confusion_df.cluster, columns=confusion_df.actual_class)
print(res)
res=res.reset_index()
a=res.idxmax(axis=1)
cluster_dict={}
for i in range(len(res)):
    cluster_dict[res['cluster'][i]]=a[i]
print(cluster_dict)
predict_tar=[cluster_dict[i] for i in predictt]
print(metrics.classification_report(test["label"], predict_tar))
```

```
actual_class    1    2    3    7
cluster
0              13    6    6  134
1             113  284  127   94
2             205   17   61   32
3               1    7  161   13
{0: 7, 1: 2, 2: 1, 3: 3}
              precision    recall  f1-score   support

           1       0.65      0.62      0.63       332
           2       0.46      0.90      0.61       314
           3       0.88      0.45      0.60       355
           7       0.84      0.49      0.62       273

    accuracy                           0.62      1274
   macro avg       0.71      0.62      0.62      1274
weighted avg       0.71      0.62      0.62      1274
```

In [12]:

```
def cluster_gmm(train, test_text, test_label):

   # Add your code
```

```
  File "C:\Users\Rossshhh\AppData\Local\Temp/ipykernel_9940/2124952698.py", line 3
    # Add your code
                   ^
IndentationError: expected an indented block
```

In [ ]:

```
cluster_gmm(train["text"], test_text, test_label)
```

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 [ ]:

```
def cluster_lda(train, test_text, test_label):

    stop_list = list(stopwords.words('english'))
    vectorizer_tf = CountVectorizer(min_df=5, stop_words=stop_list)
    tf = vectorizer_tf.fit_transform(train)
    feature_tf_names = vectorizer_tf.get_feature_names_out()

    topics = 4
    lda = LatentDirichletAllocation(n_components=topics,max_iter=30,verbose=1,evaluate_every=1, n_jobs=1,random_state=0).fit(tf)
    top_words_num=30

    for index_topic, topic in enumerate(lda.components_):
        print ("topic %d:" % (index_topic))

        words_a=[feature_tf_names[i] for i in topic.argsort()[::-1][0:top_words_num]]
        print(words_a)
        print("\n")

    test_tf = vectorizer_tf.transform(test_text)
    topic_asig=lda.transform(test_tf)
    predicted=np.argmax(topic_asig,axis=1)

    df_confu = pd.DataFrame(list(zip(test["label"].values, predicted)),columns = ["actual_class", "cluster"])
    df_confu.head()

    result=pd.crosstab( index=df_confu.cluster, columns=df_confu.actual_class)
    print(result)

    result=result.reset_index()
    a=result.idxmax(axis=1)
    cluster_dictionary={}
    for a in range(len(result)):
        cluster_dictionary[result['cluster'][a]]=a[a]
    predicted_tar=[cluster_dictionary[a] for a in predicted]
    print(metrics.classification_report(test_label, predicted_tar))
```

In [ ]:

```
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 [ ]:

```
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 [ ]:

```
```