uOttawa

ELG 5125:
Data Science Applications
Assignment 2

Prof. Arya Rahgozar

GROUP: DSA_202101_13

Table of Contents

Table of Figures

Preprocessing and data cleaning

Starting Packages

```
! pip install pyLDAvis
```

Used Libraries

```
import nltk
import pandas as pd
from sklearn.decomposition import PCA
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from nltk.probability import FreqDist
import re
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from wordcloud import WordCloud, STOPWORDS
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from gensim.models.coherencemodel import CoherenceModel
from gensim.corpora.dictionary import Dictionary
import scipy.cluster.hierarchy as sch
from scipy import stats
from sklearn.cluster import AgglomerativeClustering
from sklearn.mixture import GaussianMixture
import pyLDAvis
import gensim.corpora as corpora
from gensim.models.ldamodel import LdaModel
import pyLDAvis.gensim_models as gensimvis
from sklearn.manifold import TSNE
%matplotlib inline
```

Data Selection

Choosing "gutenburg" list of books from NLTK library as the raw data to work on.

```
nltk.download('gutenberg')
```

```
nltk.corpus.gutenberg.fileids()
```

Picking five different books from the dataset.

```
book_1 = ' '.join(nltk.corpus.gutenberg.words('austen-emma.txt'))
book_2 = ' '.join(nltk.corpus.gutenberg.words('bible-kjv.txt'))
book_3 = ' '.join(nltk.corpus.gutenberg.words('carroll-alice.txt'))
book_4 = ' '.join(nltk.corpus.gutenberg.words('edgeworth-parents.txt'))
book_5 = ' '.join(nltk.corpus.gutenberg.words('whitman-leaves.txt'))
```

```
df = pd.DataFrame({'Text':[book_1,book_2,book_3,book_4,book_5],
                   'label':['austen-emma','bible-kjv','carroll-
alice','edgeworth-parents','whitman-leaves']})
```

Data Cleaning

Cleaning the data using Regex, removing stop words and applying stemming.

```
import re
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
nltk.download('stopwords')
def clean_books(df):
  stemer=PorterStemmer()
  corpus = []
  for i in range(0,len(df)):
    # replace any character with space and leave the from (a - z )
    text = re.sub('[^A-Za-z]',' ',df['Text'][i])
    text = text.lower()
    text = text.split()
    text = [stemer.stem(word) for word in text if word not in set(stopwo
rds.words('english'))]
    text = ' '.join(text)
    corpus.append(text)
  return corpus
```

Feature Engineering

Creating Partitions

Preparing the data by creating a function named “samples” for making 200 partitions out of each book with 100 words each.

```
def samples(book):
  l=[]
  count = 0

  while count <200:
    sample = np.random.choice(book,150)
    l.append(sample)
    count+= 1
  return l
```

Frequency Distribution and Tokenization

Using frequency distribution for showing outcomes of an experiment and using sentence tokenizer “punct” on two labels.

```
nltk.download('punkt')

fdist = FreqDist(nltk.word_tokenize(df_final['Text_sample'][df_final['label']=='austen-emma'][10]))
print(fdist)
print(fdist.most_common(2))

fdist.plot(30,cumulative=False)
plt.show()
```

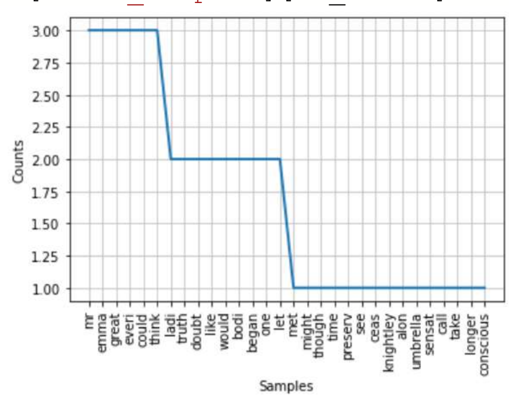


Figure 1 Frequency Distribution of "austen-emma"

```
fdist = FreqDist(nltk.word_tokenize(df_final['Text_sample'][df_final['label']=='bible-kjv'][250]))
print(fdist)
print(fdist.most_common(2))

fdist.plot(30,cumulative=False)
plt.show()
```

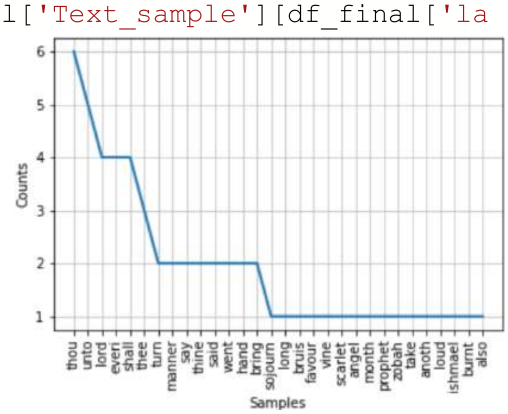


Figure 2 Frequency Distribution of "bible-kjv"

Word Cloud

Creating a function to create word cloud of any book.

```
def worldcloud(df):
  comment_words = ''
  stopwords = set(STOPWORDS)
  # iterate through the csv file
  for val in df:
      # typecaste each val to string
      val = str(val)
      # split the value
      tokens = val.split()
      # Converts each token into lowercase
      for i in range(len(tokens)):
          tokens[i] = tokens[i].lower()
      comment_words += " ".join(tokens)+" "
  wordcloud = WordCloud(width = 800, height = 800,
                        background_color ='white',
                        stopwords = stopwords,
                        min_font_size = 10).generate(comment_words)
  # plot the WordCloud image
  plt.figure(figsize = (8, 8), facecolor = None)
  plt.imshow(wordcloud)
  plt.axis("off")
  plt.tight_layout(pad = 0)
  plt.show()
```

Then applying the function on all of the five books to show most frequent words.

```
worldcloud(emma)
```

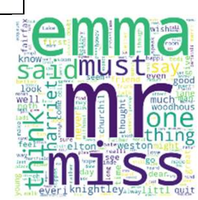


Figure 3 Word cloud of book 1

```
bible_kjv = df_final['Text_sample'][df_final['label']==
'bible-kjv']
worldcloud(bible_kjv)
```

Figure 4 Word cloud of book 2

```
carroll_alice = df_final['Text_sample'][df_final['label']==
'carroll-alice']
worldcloud(carroll_alice)
```

Figure 5 Word cloud of book 3

```
edgeworth_parents = df_final['Text_sample'][df_final['label']==
'edgeworth-parents']
worldcloud(edgeworth_parents)
```

Figure 7 Word cloud of book 4

```
whitman_leaves = df_final['Text_sample'][df_final['label']==
'whitman-leaves']
worldcloud(whitman_leaves)
```

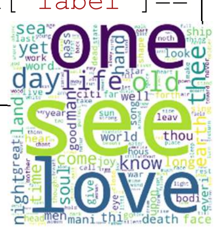


Figure 6 Word cloud of book 5

Modelling

K-means

BOW

```
bow = CountVectorizer()
X_bow = bow.fit_transform(X).toarray()
pd.DataFrame(X_bow,columns=bow.get_feature_names())
```

```
# Elbow Method
wcss = []
for i in range(1, 20):
    kmeans = KMeans(n_clusters=i,
init='k-means++', random_state=0)
    kmeans.fit(X_bow)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 20), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

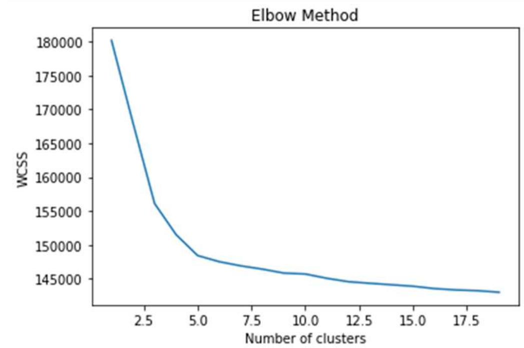


Figure 8 Elbow method for bag of words

N.B: It is clear that the best number for K is 5

```
# Clusters Visualization
tsne = TSNE(n_components=2, verbose=1, random_state=42)
z = tsne.fit_transform(X_bow)

tsne_df = pd.DataFrame()
tsne_df["y"] = y
tsne_df["comp-1"] = z[:,0]
tsne_df["comp-2"] = z[:,1]

sns.scatterplot(x="comp-1", y="comp-2", hue=tsne_df.y.tolist(),
                palette=sns.color_palette("hls", 5),
                data=tsne_df).set(title="Kmean T-SNE projection")
```

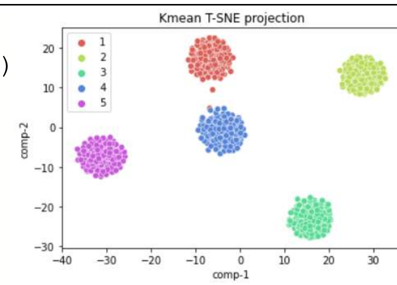


Figure 9 T-SNE clustering for BOW

```
# Error analysis
if kappa_score != 1:
  error_prediction=compined_data[
compined_data['cluster_name']!= compined_data['label']]

  error_text = error_prediction['Text_sample']
  worldcloud(error_text)
else:
  print('We can not plot word cloud
 for wordes here becuse no error her, Kappa score = 1.0')
```

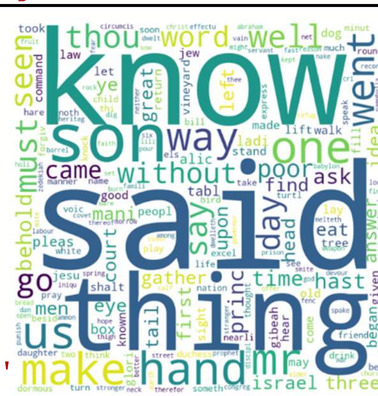


Figure 10 Word Cloud for false predictions in k-means, BOW

BOW with 2000 feature

```
bow= CountVectorizer(max_features= 2000)
X_bow_2000 = bow.fit_transform(X).toarray()
```

```
pd.DataFrame(X_bow_2000,columns=bow.get_feature_names())
```

```
# Elbow Method
wcss = []
for i in range(1, 20):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=0)
    kmeans.fit(X_bow_2000)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 20), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

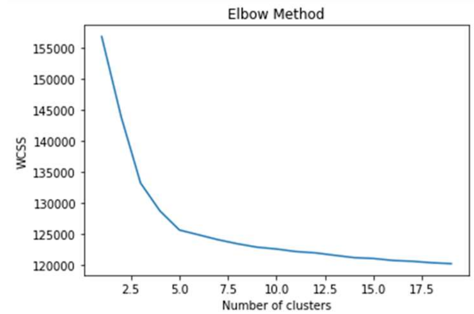


Figure 11 Elbow method for BOW for 2000 features

N.B: It is clear that the best number for K is 5

```
# Clusters Visualization
tsne = TSNE(n_components=2, verbose=1, random_state=42)
z = tsne.fit_transform(X_bow_2000)

tsne_df = pd.DataFrame()
tsne_df["y"] = pred_y_2000
tsne_df["comp-1"] = z[:,0]
tsne_df["comp-2"] = z[:,1]

sns.scatterplot(x="comp-1", y="comp-2", hue=tsne_df.y.tolist(),
                palette=sns.color_palette("hls", 5),
                data=tsne_df).set(title="kmeans 2000 T-SNE projection")
```

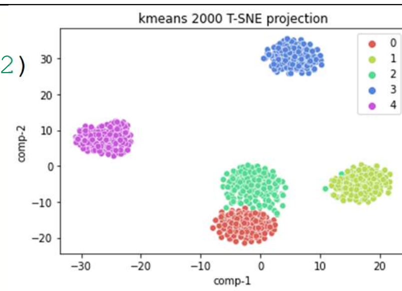


Figure 12 T-SNE clustering for K-means, BOW for 2000 features

```
# Error analysis
if kappa_score != 1:
  error_prediction=compined_data[compined_data
['cluster_name_for_2000_bw']!= compined_data['label']]

  error_text = error_prediction['Text_sample']
  worldcloud(error_text)
else:
  print('We can not plot word
cloud for wordes here becuse no error her,
 Kappa score = 1.0')
```

Figure 13 Word cloud for false predictions in K-means, BOW for 2000 features

TF-IDF

```
tf = TfidfVectorizer()
X_tf=tf.fit_transform(X).toarray()
pd.DataFrame(X_tf,columns=tf.get_feature_names())
```

```
# Elbow Method
wcss = []
for i in range(1, 20):
    kmeans = KMeans(n_clusters=i, init='k-
means++', random_state=0)
    kmeans.fit(X_tf)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 20), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

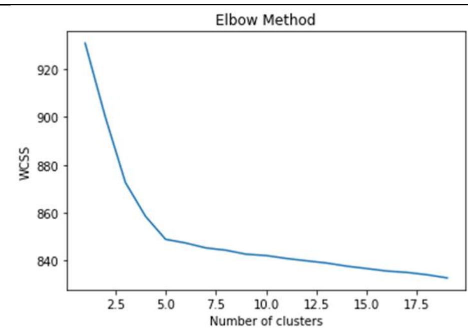


Figure 14 Elbow method for TF-IDF

N.B: It is clear that the best number for K is 5

```
# Clusters Visualization
tsne = TSNE(n_components=2, verbose=1, random_state=42)
z = tsne.fit_transform(X_tf)

tsne_df = pd.DataFrame()
tsne_df["y"] = pred_y_tf
tsne_df["comp-1"] = z[:,0]
tsne_df["comp-2"] = z[:,1]

sns.scatterplot(x="comp-1", y="comp-2", hue=tsne_df.y.tolist(),
                palette=sns.color_palette("hls", 5),
                data=tsne_df).set(title="kmeans TF T-SNE projection")
```

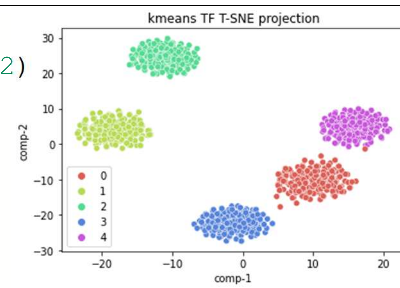


Figure 15 T-SNE clustering for K-means, TF-IDF

```
# Error analysis
if kappa_score != 1:
  error_prediction=compined_data[compined_data['cluster_name']!=
compined_data['label']]

  error_text = error_prediction['Text_sample']
  worldcloud(error_text)
else:
  print('We can not plot word cloud for
 wordes here becuse no error her, Kappa score = 1.0')
```

N.B: No plot was shown as it had no error (kappa score = 1.0)

TF-IDF with Bi-gram

```
tf = TfidfVectorizer(ngram_range=(2,2))
X_tf_bi=tf.fit_transform(X).toarray()
```

```
pd.DataFrame(X_tf_bi,columns=tf.get_feature_names())
```

```
# Elbow Method
wcss = []
for i in range(1, 20):
    kmeans = KMeans(n_clusters=i, init='k-
means++', random_state=0)
    kmeans.fit(X_tf_bi)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 20), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

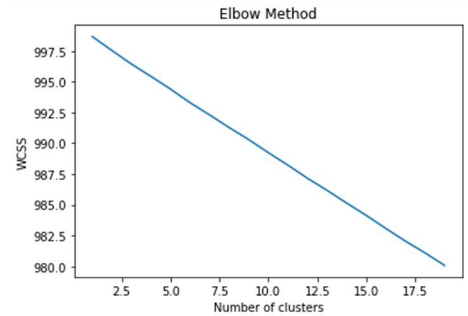


Figure 16 Elbow method for TF-IDF with Bi-gram

```
# Clusters Visualization
tsne = TSNE(n_components=2, verbose=1, random_state=42)
z = tsne.fit_transform(X_tf_bi)

tsne_df = pd.DataFrame()
tsne_df["y"] = pred_y_tf_bi
tsne_df["comp-1"] = z[:,0]
tsne_df["comp-2"] = z[:,1]

sns.scatterplot(x="comp-1", y="comp-2", hue=tsne_df.y.tolist(),
                palette=sns.color_palette("hls", 5),
                data=tsne_df).set(title="kmeans TF-BI T-SNE projection")
```

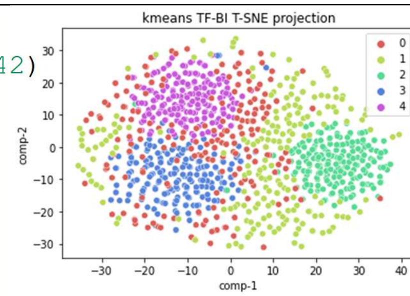


Figure 17 T-SNE clustering for K-means, TF-IDF with Bi-gram

```
# Error analysis
if kappa_score != 1:
  error_prediction=compined_data[compined_data
['cluster_name']!= compined_data['label']]

  error_text = error_prediction['Text_sample']
  worldcloud(error_text)
else:
  print('We can not plot word cloud for
 wordes here becuse no error her, Kappa score = 1.0')
```

N.B: No plot was shown as it had no error (kappa score = 1.0)

Agglomerative

BOW

```
bow = CountVectorizer()
X_bow = bow.fit_transform(X).toarray()
pd.DataFrame(X_bow,columns=bow.get_feature_names())
```

```
# Elbow Method
wcss = []
for i in range(1, 20):
    kmeans = KMeans(n_clusters=i,
init='k-means++', random_state=0)
    kmeans.fit(X_bow)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 20), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

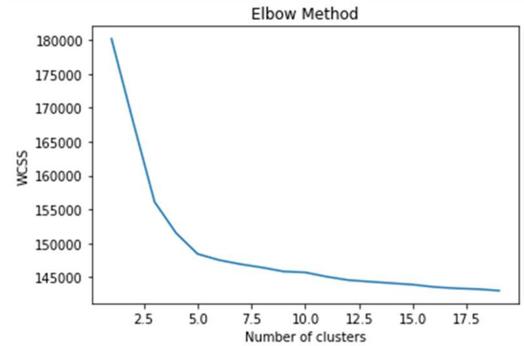


Figure 18 Elbow method for bag of words

N.B: It is clear that the best number for K is 5

```
# Clusters Visualization
tsne = TSNE(n_components=2, verbose=1, random_state=42)
z = tsne.fit_transform(X_bow)

tsne_df = pd.DataFrame()
tsne_df["y"] = y_hc
tsne_df["comp-1"] = z[:,0]
tsne_df["comp-2"] = z[:,1]

sns.scatterplot(x="comp-1", y="comp-2", hue=tsne_df.y.tolist(),
                palette=sns.color_palette("hls", 5),
                data=tsne_df).set(title="Agglomerative T-SNE projection")
```

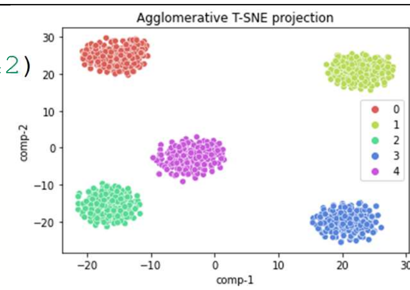


Figure 19 T-SNE clustering for agglomerative, BOW

```
# Error analysis
if kappa_score != 1:
  error_prediction=compined_data[compined_data
['cluster_name_agglomerative']!= compined_data['label']]

  error_text = error_prediction['Text_sample']
  worldcloud(error_text)
else:
  print('We can not plot word cloud for words
 here becuse no error her, Kappa score = 1.0')
```

N.B: No plot was shown as it had no error (kappa score = 1.0)

BOW with 2000 features

```
bow= CountVectorizer(max_features= 2000)
X_bow_2000 = bow.fit_transform(X).toarray()
pd.DataFrame(X_bow_2000,columns=bow.get_feature_names())
```

```
# Elbow Method
wcss = []
for i in range(1, 20):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=0)
    kmeans.fit(X_bow_2000)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 20), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

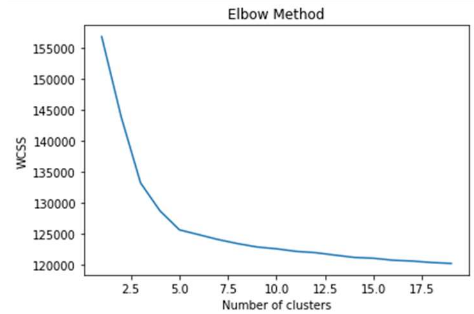


Figure 20 Elbow method for BOW with 2000 features

N.B: It is clear that the best number for K is 5

```
# Clusters Visualization
tsne = TSNE(n_components=2, verbose=1, random_state=42)
z = tsne.fit_transform(X_bow_2000)

tsne_df = pd.DataFrame()
tsne_df["y"] = y_hc_2000
tsne_df["comp-1"] = z[:,0]
tsne_df["comp-2"] = z[:,1]

sns.scatterplot(x="comp-1", y="comp-2", hue=tsne_df.y.tolist(),
                palette=sns.color_palette("hls", 5),
                data=tsne_df).set(title="Agglomerative 2000 T-SNE projection")
```

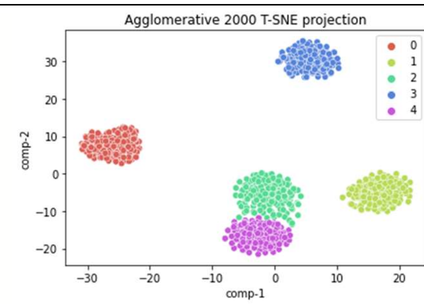


Figure 21 T-SNE clustering for agglomerative, BOW with 2000 features

```
# Error analysis
if kappa_score != 1:
  error_prediction=compined_data[compined_data['cluster_agglomerative_clustering_2000_name']!= compined_data['label']]

  error_text = error_prediction['Text_sample']
  worldcloud(error_text)
else:
  print('We can not plot word cloud for words here becuse no error her, Kappa score = 1.0')
```

N.B: No plot was shown as it had no error (kappa score = 1.0)

TF-IDF

```
tf = TfidfVectorizer()
X_tf=tf.fit_transform(X).toarray()
```

```
pd.DataFrame(X_tf,columns=tf.get_feature_names())
```

```
# Elbow Method
wcss = []
for i in range(1, 20):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=0)
    kmeans.fit(X_tf)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 20), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

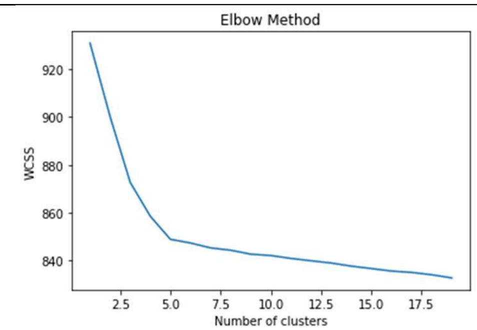


Figure 22 Elbow method for TF-IDF

N.B: It is clear that the best number for K is 5

```
# Clusters Visualization
tsne = TSNE(n_components=2, verbose=1, random_state=42)
z = tsne.fit_transform(X_tf)

tsne_df = pd.DataFrame()
tsne_df["y"] = y_hc_tf
tsne_df["comp-1"] = z[:,0]
tsne_df["comp-2"] = z[:,1]

sns.scatterplot(x="comp-1", y="comp-2", hue=tsne_df.y.tolist(),
                palette=sns.color_palette("hls", 5),
                data=tsne_df).set(title="Agglomerative TF T-SNE projection")
```

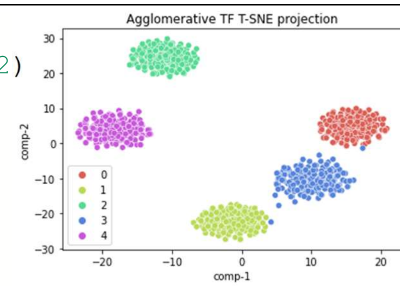


Figure 23 T-SNE clustering for agglomerative, TF-IDF

```
# Error analysis
if kappa_score != 1:
  error_prediction=compined_data[compined_data['agg_cluster_tf_name']!= compined_data['label']]

  error_text = error_prediction['Text_sample']
  worldcloud(error_text)
else:
  print('We can not plot word cloud for words here becuse no error her, Kappa score = 1.0')
```

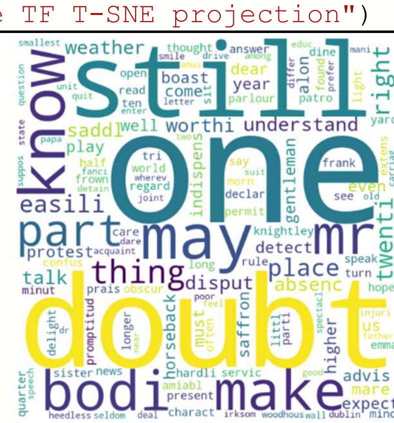


Figure 24 Word cloud for false predictions in agglomerative, TF-IDF

TF-IDF with Bi-gram

```
tf = TfidfVectorizer(ngram_range=(2,2))
X_tf_bi=tf.fit_transform(X).toarray()
```

```
pd.DataFrame(X_tf_bi,columns=tf.get_feature_names())
```

```
# Elbow Method
wcss = []
for i in range(1, 20):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=0)
    kmeans.fit(X_tf_bi)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 20), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

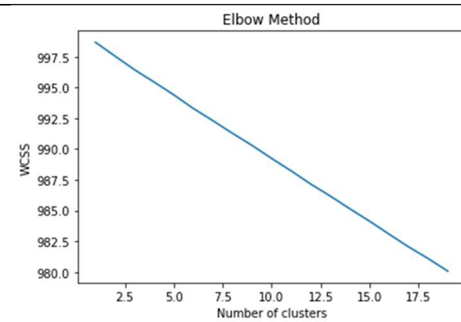


Figure 25 Elbow method for TF-IDF with Bi-gram

```
# Clusters Visualization
tsne = TSNE(n_components=2, verbose=1, random_state=42)
z = tsne.fit_transform(X_tf_bi)

tsne_df = pd.DataFrame()
tsne_df["y"] = y_hc_tf_bi
tsne_df["comp-1"] = z[:,0]
tsne_df["comp-2"] = z[:,1]

sns.scatterplot(x="comp-1", y="comp-2", hue=tsne_df.y.
                palette=sns.color_palette("hls", 5),
                data=tsne_df).set(title="Agglomerative TF-BI T-SNE projection")
```

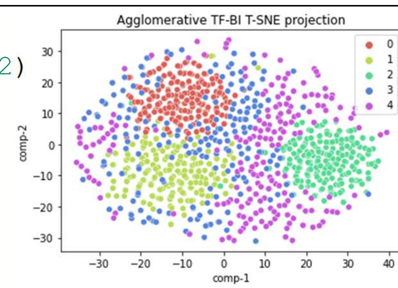


Figure 26 T-SNE clustering for agglomerative, TF-IDF with Bi-Gram

```
# Error analysis
if kappa_score != 1:
  error_prediction=compined_data[compined_data['agg_cluster_tf_bi_name']!= compined_data['label']]
  error_text = error_prediction['Text_sample']
  worldcloud(error_text)
else:
  print('We can not plot word cloud for words here becuse no error her, Kappa score = 1.0')
```

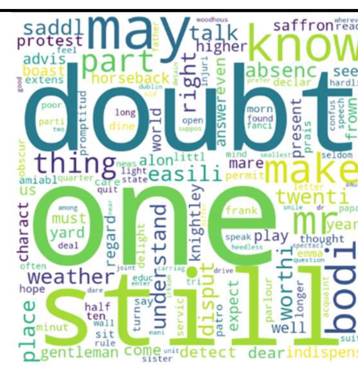


Figure 27 Word cloud for false predictions in agglomerative, TF-IDF with Bi-Gram

Gaussian

BOW

```
bow = CountVectorizer()
X_bow = bow.fit_transform(X).toarray()
pd.DataFrame(X_bow,columns=bow.get_feature_names())
```

```
# Elbow Method
wcss = []
for i in range(1, 20):
    kmeans = KMeans(n_clusters=i,
init='k-means++', random_state=0)
    kmeans.fit(X_bow)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 20), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

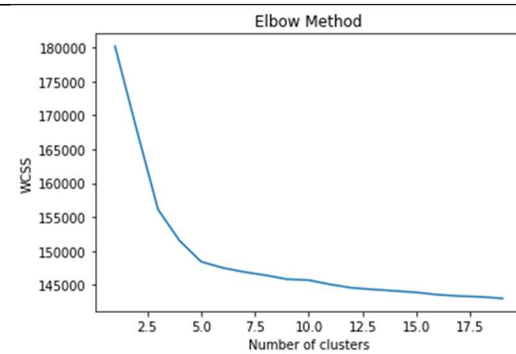


Figure 28 Elbow method for Bag of words

N.B: It is clear that the best number for K is 5

```
# Clusters Visualization
tsne = TSNE(n_components=2, verbose=1, random_state=42)
z = tsne.fit_transform(X_bow)

tsne_df = pd.DataFrame()
tsne_df["y"] = y
tsne_df["comp-1"] = z[:,0]
tsne_df["comp-2"] = z[:,1]

sns.scatterplot(x="comp-1", y="comp-2", hue=tsne_df.y.tolist(),
                palette=sns.color_palette("hls", 5),
                data=tsne_df).set(title="Gaussian Mixture T-SNE projection")
```

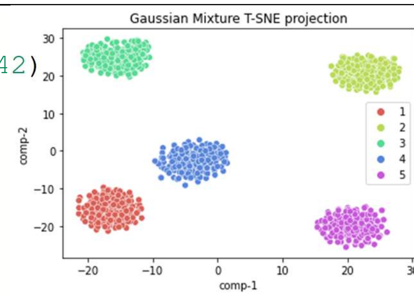


Figure 29 T-SNE clustering for BOW

```
# Error analysis
if kappa_score != 1:
  error_prediction=compined_data[compined_data
['cluster_name']!= compined_data['label']]

  error_text = error_prediction['Text_sample']
  worldcloud(error_text)
else:
  print('We can not plot word cloud for words
here becuse no error her, Kappa score = 1.0')
```

Figure 30 Word cloud for false predictions in Gaussian, BOW

BOW with 2000 feature

```
bow= CountVectorizer(max_features= 2000)
X_bow_2000 = bow.fit_transform(X).toarray()
```

```
pd.DataFrame(X_bow_2000,columns=bow.get_feature_names())
```

```
# Elbow Method
wcss = []
for i in range(1, 20):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=0)
    kmeans.fit(X_bow_2000)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 20), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

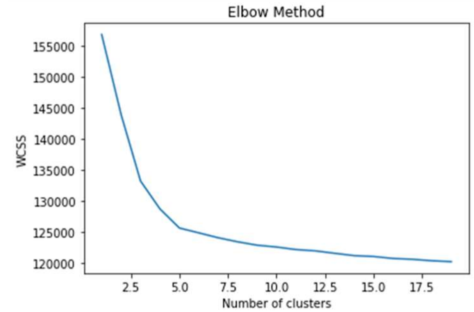


Figure 31 Elbow method for BOW with 2000 features

N.B: It is clear that the best number for K is 5

```
# Clusters Visualization
tsne = TSNE(n_components=2, verbose=1, random_state=42)
z = tsne.fit_transform(X_bow_2000)

tsne_df = pd.DataFrame()
tsne_df["y"] = labels_2000
tsne_df["comp-1"] = z[:,0]
tsne_df["comp-2"] = z[:,1]

sns.scatterplot(x="comp-1", y="comp-2", hue=tsne_df.y.tolist(),
                palette=sns.color_palette("hls", 5),
                data=tsne_df).set(title="Gaussian Mixture 2000 T-SNE projection")
```

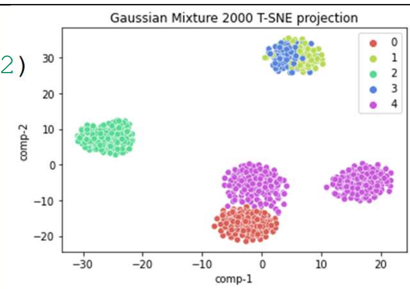


Figure 32 T-SNE clustering for gaussian, BOW with 2000 features

```
# Error analysis
if kappa_score != 1:
  error_prediction=compined_data[compined_data
 ['gmm_2000_name']!= compined_data['label']]

  error_text = error_prediction['Text_sample']
  worldcloud(error_text)
else:
  print('We can not plot word cloud for words
 here becuse no error her, Kappa score = 1.0')
```

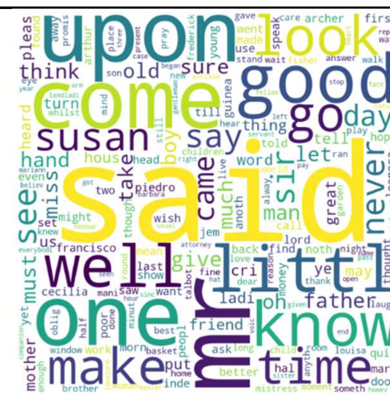


Figure 33 Word cloud for false predictions for gaussian, BOW with 2000 features

TF-IDF with PCA

```
tf = TfidfVectorizer()
X_tf=tf.fit_transform(X).toarray()
```

```
pd.DataFrame(X_tf,columns=tf.get_feature_names())
```

```
# Elbow Method
wcss = []
for i in range(1, 20):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=0)
    kmeans.fit(X_tf)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 20), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

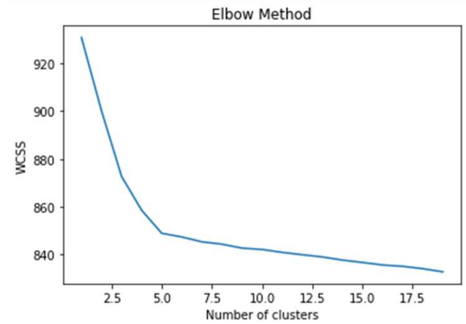


Figure 34 Elbow method for TF-IDF

N.B: It is clear that the best number for K is 5

```
# Clusters Visualization
tsne = TSNE(n_components=2, verbose=1, random_state=42)
z = tsne.fit_transform(X_bow)

tsne_df = pd.DataFrame()
tsne_df["y"] = y
tsne_df["comp-1"] = z[:,0]
tsne_df["comp-2"] = z[:,1]

sns.scatterplot(x="comp-1", y="comp-2", hue=tsne_df.y.tolist(),
                palette=sns.color_palette("hls", 5),
                data=tsne_df).set(title="GaussianMixture TF T-SNE projection")
```

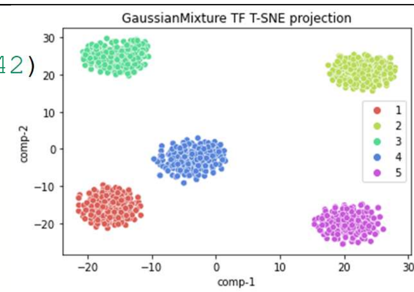


Figure 35 T-SNE clustering for Gaussian, TF-IDF

```
# Error analysis
if kappa_score != 1:
  error_prediction=compined_data[compined_data['cluster_name_tf']!= compined_data['label']]

  error_text = error_prediction['Text_sample']
  worldcloud(error_text)
else:
  print('We can not plot word cloud for words here becuse no error her, Kappa score = 1.0')
```

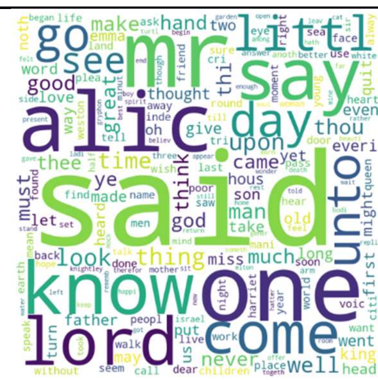


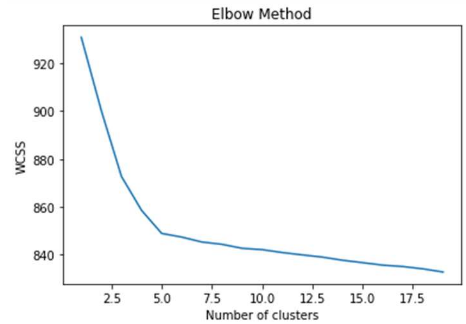
Figure 36 Word cloud for false predictions for Gaussian, TF-IDF

TF-IDF with Bi-gram and PCA

```
X_tf_bi_pca=PCA(n_components=2)
X_tf_bi_pca.fit(X_tf_bi)
```

```
gmm_tf_bi = GaussianMixture(n_components=5)
gmm_tf_bi.fit(X_tf_bi)
labels_tf_bi = gmm_tf.predict(X_tf_bi)
```

```
# Elbow Method
wcss = []
for i in range(1, 20):
    kmeans = KMeans(n_clusters=i, init='k-
means++', random_state=0)
    kmeans.fit(X_tf)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 20), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



N.B: It is clear that the best number for K is 5

```
# Clusters Visualization
tsne = TSNE(n_components=2, verbose=1, random_state=42)
z = tsne.fit_transform(X_tf_bi)

tsne_df = pd.DataFrame()
tsne_df["y"] = pred_y_tf_bi
tsne_df["comp-1"] = z[:,0]
tsne_df["comp-2"] = z[:,1]

sns.scatterplot(x="comp-1", y="comp-2", hue=tsne_df.y.tolist(),
                palette=sns.color_palette("hls", 5),
                data=tsne_df).set(title="gmm  TF-BI T-SNE projection")
```

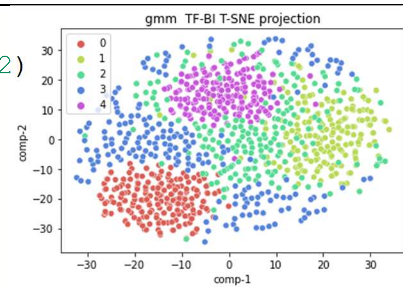


Figure 37 T-SNE clustering for Gaussian, TF-IDF Bi-gram with PCA

```
# Error analysis
if kappa_score != 1:
  error_prediction=compined_data[compined_data
['cluster_name_tf']!= compined_data['label']]

  error_text = error_prediction['Text_sample']
  worldcloud(error_text)
else:
  print('We can not plot word cloud for words
here becuse no error her, Kappa score = 1.0')
```

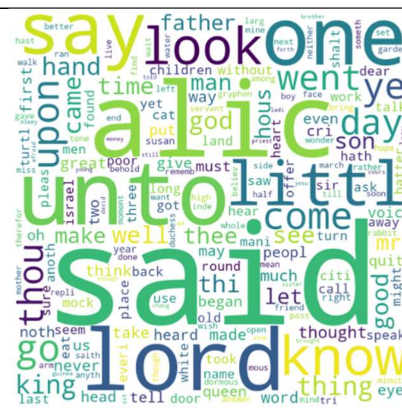


Figure 38 Word cloud for false predictions for Gaussian, TF-IDF with PCA

Applying LDA

```
from pycaret.utils import enable_colab
enable_colab()
```

```
from pycaret.nlp import *
```

```
lda = setup(data = df_final, target = 'Text_sample', session_id=123)
```

```
m1 = create_model(model='lda',num_topics = 5, multi_core=True)
```

```
lda_data = assign_model(m1)
```

```
lda_data.head()
```

	Text_sample_	label	Text_sample	Topic_0	Topic_1	Topic_2	Topic_3	Topic_4	Dominant_Topic	Perc_Dominant_Topic
0	['knightley' 'difficulti' 'feel' 'polit' 'disp...	austen-emma	difficulti feel polit display gentleman notic ...	0.002235	0.916281	0.002211	0.077083	0.002190	Topic 1	0.92
1	['brought' 'manner' 'endeavour' 'afraid' 'turn...	austen-emma	bring manner turn would bring seem amiabl thin...	0.002795	0.988797	0.002800	0.002797	0.002811	Topic 1	0.99
2	['place' 'pain' 'emma' 'introduc' 'would' 'loo...	austen-emma	place pain emma introduc would look welch must...	0.002124	0.991492	0.002127	0.002125	0.002131	Topic 1	0.99
3	['occur' 'world' 'even' 'without' 'convinc' 'u...	austen-emma	occur world even convinc understand cottag wou...	0.002212	0.925357	0.068038	0.002204	0.002190	Topic 1	0.93
4	['reflect' 'suspici' 'practis' 'without' 'enou...	austen-emma	reflect practis enough woman feel woman want p...	0.002862	0.988456	0.002891	0.002876	0.002915	Topic 1	0.99

Figure 39 Data with LDA

```
lda_data.Dominant_Topic .value_counts()
```

Figure 40 Dominant topics in LDA

```
Topic 2    417
Topic 4    236
Topic 1    191
Topic 3    108
Topic 0     48
Name: Dominant_Topic, dtype: int64
```

```
lda_model = LdaModel(corpus=corpus,
                     id2word=id2word,
                     num_topics=5,
                     random_state=0,
                     chunksize=100,
                     alpha='auto',
                     per_word_topics=True)
print(lda_model.print_topics())
doc_lda = lda_model[corpus]
```

```
coherence_model_lda = CoherenceModel(model=lda_model, texts=texts, dicti
onary=id2word, coherence='c_v')
coherence_lda = coherence_model_lda.get_coherence()
print(coherence_lda)
```

```
pyLDAvis.enable_notebook()
p = gensimvis.prepare(lda_model, corpus, id2word)
p
```

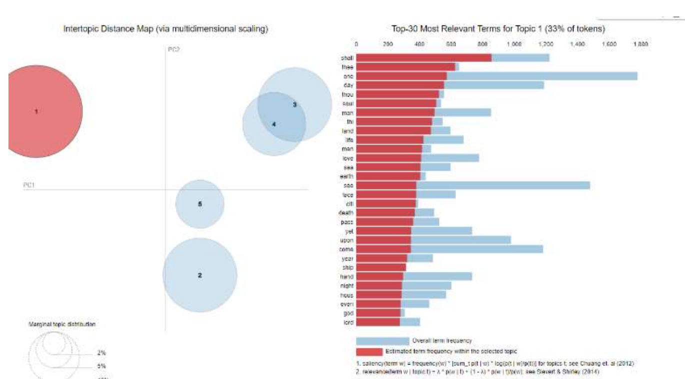


Figure 41 LDA analysis - topic 1

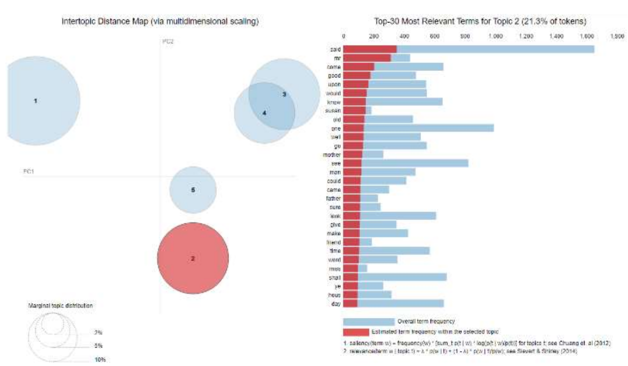


Figure 43 LDA analysis - topic 2

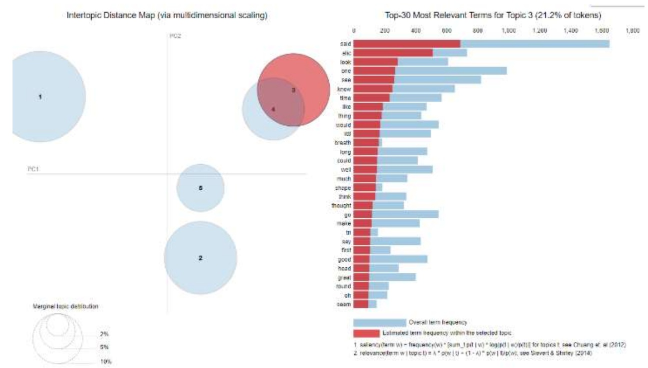


Figure 42 LDA analysis - topic 3

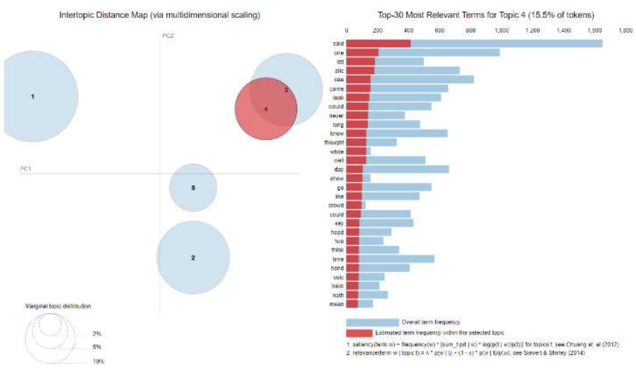


Figure 45 LDA analysis - topic 4

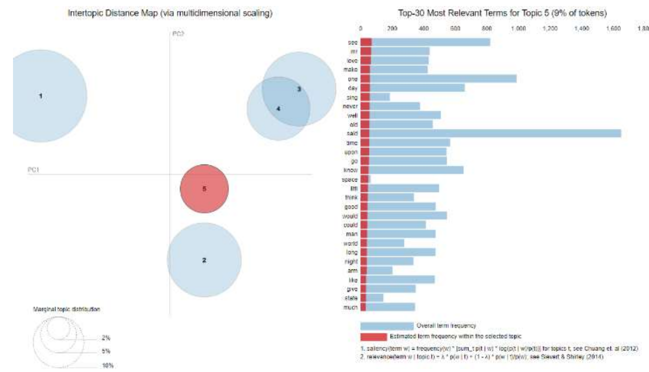


Figure 44 LDA analysis - topic 5

```
plot_model(m1, plot = 'sentiment')
```

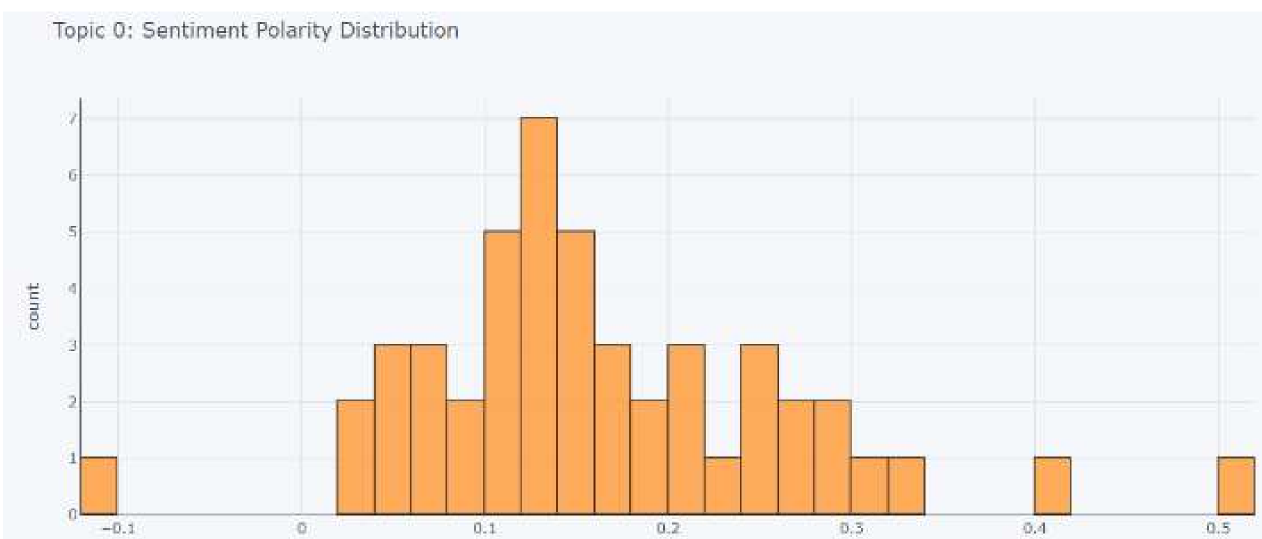


Figure 46 sentiment analysis after LDA

```
plot_model(m1, plot = 'frequency')
```

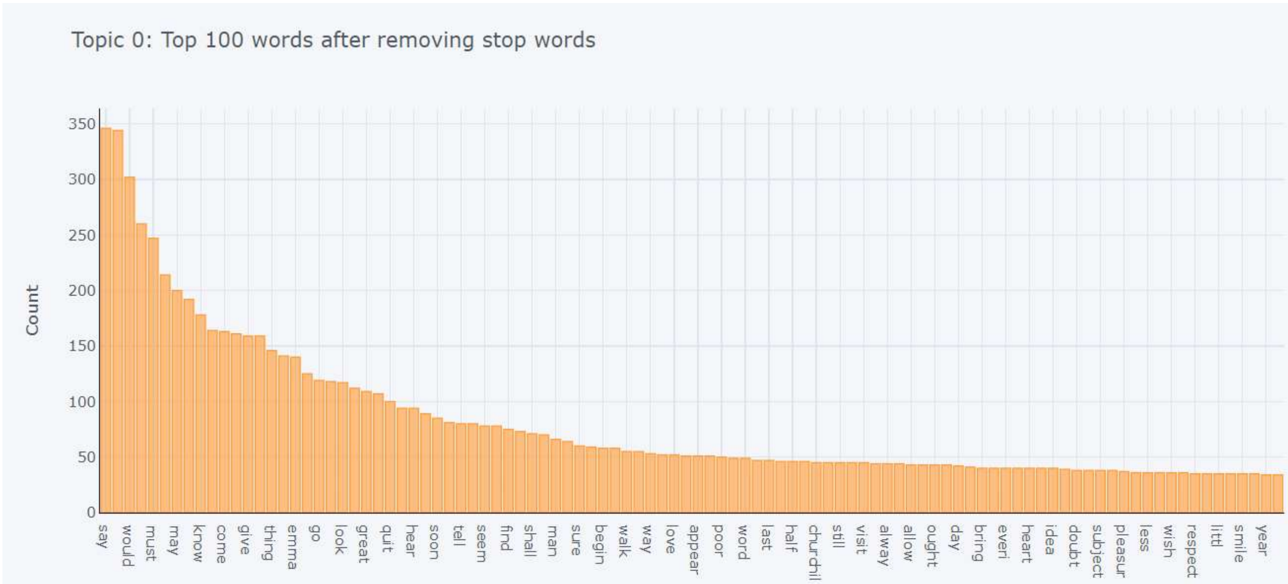


Figure 47 Top words after removing stop words

Dendrogram Visualization

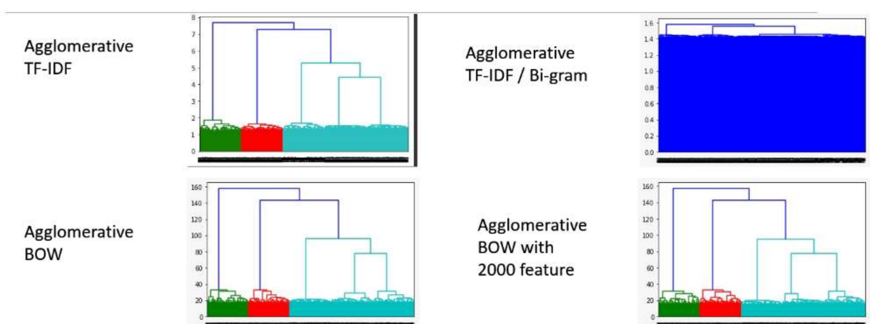


Figure 48 Dendrogram Visualization

Champion model: K-means with LDA

```
kmean_results = assign_model(kmeans)
kmean_results
```

	Topic_0	Topic_1	Topic_2	Topic_3	Topic_4	Cluster
0	0.002130	0.991427	0.002177	0.002130	0.002138	Cluster 3
1	0.002204	0.458459	0.534937	0.002218	0.002181	Cluster 3
2	0.002209	0.845739	0.147630	0.002213	0.002209	Cluster 3
3	0.129335	0.676752	0.189455	0.002218	0.002240	Cluster 3
4	0.002564	0.746186	0.246107	0.002565	0.002577	Cluster 3
...	...	...	...	...	...	...
995	0.002533	0.114544	0.002520	0.230368	0.650035	Cluster 2
996	0.003280	0.003273	0.003323	0.791370	0.198754	Cluster 4
997	0.002425	0.002472	0.002448	0.990209	0.002446	Cluster 4
998	0.002828	0.002845	0.002848	0.988623	0.002856	Cluster 4
999	0.003064	0.003031	0.039409	0.476664	0.477832	Cluster 4

1000 rows × 6 columns

Figure 49 K-means results with LDA

```
kmean_results['Cluster']
```

Figure 50 K-means with LDA clustering

```
Cluster 0    408
Cluster 2    227
Cluster 3    200
Cluster 4    116
Cluster 1     49
Name: Cluster, dtype: int64
```

```
error_prediction=compined_data[compined_data['cluster_name']!= compined_data['label']]
```

```
error_text = error_prediction['Text_sample']
worldcloud(error_text)
```

Figure 51 K-means with LDA false prediction word cloud

```
plot_model(kmeans, plot = 'elbow')
```

So the K=5 for the final k-means model

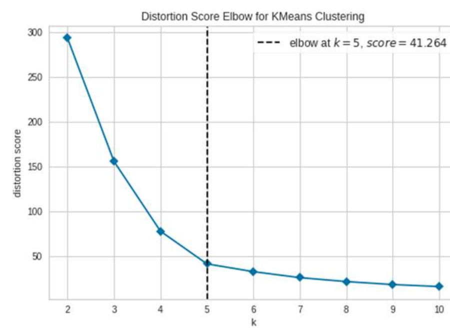


Figure 52 Elbow method for final K-means

```
plot_model(kmeans, plot = 'silhouette')
```

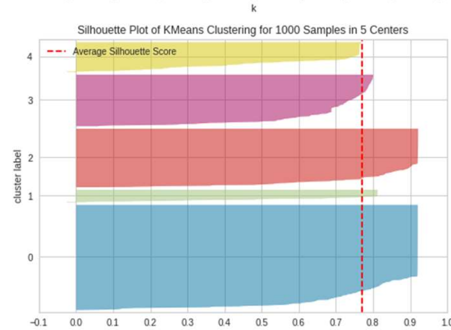


Figure 53 Silhouette plot for final K-means

Then we combine the data

```
compined_data = pd.concat([kmean_results,df_final],axis = 1)
compined_data
```

	Topic_0	Topic_1	Topic_2	Topic_3	Topic_4	Cluster	Text_sample_	label	Text_sample
0	0.002130	0.991427	0.002177	0.002130	0.002138	Cluster 3	['great' 'attend' 'say' 'long' 'moment' 'world...	austen-emma	great attend say long moment world cannot must...
1	0.002204	0.458459	0.534937	0.002218	0.002181	Cluster 3	['inde' 'mr' 'otherwis' 'mutual' 'see' 'emma' ...	austen-emma	inde mr otherwis mutual see emma met could rea...
2	0.002209	0.845739	0.147630	0.002213	0.002209	Cluster 3	['strike' 'thought' 'could' 'upon' 'ever' 'rev...	austen-emma	strike thought could upon ever reveal observ l...
3	0.129335	0.676752	0.189455	0.002218	0.002240	Cluster 3	['wretch' 'knightley' 'town' 'would' 'toward' ...	austen-emma	wretch knightley town would toward found two d...
4	0.002564	0.746186	0.246107	0.002565	0.002577	Cluster 3	['degre' 'return' 'least' 'last' 'whenev' 'emm...	austen-emma	degre return least last whenev emma darl atten...
...	...	...	...	...	...	...	...	...	...
995	0.002533	0.114544	0.002520	0.230368	0.650035	Cluster 2	['eve' 'roam' 'offend' 'farther' 'file' 'dwell...	whitman-leaves	eve roam offend farther file dwell persuad sil...
996	0.003280	0.003273	0.003323	0.791370	0.198754	Cluster 4	['serv' 'england' 'lick' 'old' 'utter' 'pass' ...	whitman-leaves	serv england lick old utter pass life centuri ...
997	0.002425	0.002472	0.002448	0.990209	0.002446	Cluster 4	['vow' 'alleghani' 'beard' 'grass' 'directli' ...	whitman-leaves	vow alleghani beard grass directli curiou atla...
998	0.002828	0.002845	0.002848	0.988623	0.002856	Cluster 4	['matter' 'vision' 'ever' 'still' 'bodi' 'drea...	whitman-leaves	matter vision ever still bodi dream follow flo...
999	0.003064	0.003031	0.039409	0.476664	0.477832	Cluster 4	['utter' 'forth' 'unconsci' 'hastili' 'determi...	whitman-leaves	utter forth unconsci hastili determin stern pe...

1000 rows × 9 columns

Figure 54 Combined data of champion model

Overall evaluation for the models

	Kappa_Score	silhouette_score	coherence_score
kmeans_pow	1.00000	0.062691	0.620658
AgglomerativeClustering_pow	0.25000	0.062411	0.620658
GaussianMixture_pow	1.00000	0.062691	0.620658
kmeans_pow_2000	1.00000	0.063068	0.620658
AgglomerativeClustering_pow_2000	0.99625	0.062696	0.620658
GaussianMixture_pow_2000	1.00000	0.063068	0.620658
kmeans_tf	1.00000	0.044823	0.620658
AgglomerativeClustering_tf	1.00000	0.044823	0.620658
GaussianMixture_tf	0.25000	0.044823	0.620658
kmeans_tf_bi	1.00000	0.000336	0.620658
AgglomerativeClustering_tf_bi	0.25000	-0.000050	0.620658
GaussianMixture_tf_bi	0.25000	-0.000050	0.620658

Conclusion

Finally, we built three models to complete this task. These models are K-means, Agglomerative Clustering, and Gaussian Mixture. We applied each model on four different feature engineering techniques such as Bag of Word (BOW), Bag of Word (BOW) with best 2000 features, Term Frequency-Inverse Document Frequency (TF-IDF), Term Frequency-Inverse Document Frequency (TF-IDF) with bigram. Then to evaluate each model we calculate Kappa, silhouette, coherence scores for each one. After that we applied Latent Dirichlet Allocation (LDA) as feature engineering with k-means and achieved the high silhouette score near to one that`s mean the distance between the clusters is high. At last, we applied some error analyses upon each model with each feature engineering step to show the text that each model makes error clustering on it. and we used t-distributed Stochastic Neighbor Embedding (t-SNE) to visualize clusters in 2D-dimentions.