DTI5125[EG] Data Science Applications 2021

Group Assignment (2)

Under Supervision:

Prof. Dr. Arya Rahgozar

Group Members:

Abdelrahman Basha

Hussien Abdelraouf

Dina Ibrahim

Table of contents:

Abstract:

This report aims to produce is to produce similar clusters and compare them; analyze the pros and cons of algorithms, generate and communicate the insights and discuss the implementation steps of applying (preprocessing the data, transformation and evaluation,etc) strategies on three of Clustering algorithms (K-means, EM, Hierarchical clustering algorithm). We will introduce the detailed steps of implementation of this strategy and discuss the achieved results. We used (five different samples of Gutenberg digital books) for implementation.

Introduction:

Text Clustering is the task of partitioning the dataset into groups called clusters. The goal is to split up the data in such a way that points within single cluster are very similar and points in different clusters are different. It determines grouping among unlabeled data. Text clustering algorithms process text and determine if natural clusters (groups) exist in the data.

Methodology:

We prepared the data by creating random samples of 200 documents per book and limited them to 150 words. We preprocessed the data by removing stop words, doing stemming and lemmatization. We transformed it to BOW, TF-IDF, N-gram and LDA. We binarized the values to use them with the K-means, EM, Hierarchical clustering algorithms to calculate Kappa against true authors, Coherence and Silhouette. We Compared and decided which clustering result is the closest to the human labels Finally, we did an Error-Analysis for the champion model to identity what were the characteristics of the instance records that threw the machine off.

Gutenberg project :

What is Gutenberg digital books

Project Gutenberg is a library of over 60,000 free eBooks.

Project Gutenberg (PG) is a volunteer effort to digitize and archive cultural works, as well as to "encourage the creation and distribution of eBooks." It was founded in 1971 by American

writer Michael S. Hart and is the oldest digital library. Most of the items in its collection are the full texts of books in the public domain. The Project tries to make these as free as possible, in long-lasting, open formats that can be used on almost any computer. As of 22 May 2021, Project Gutenberg had reached 65,405 items in its collection of free eBooks.

Gutenberg project features:

Project Gutenberg is zealously noncommercial, digitizes books in the public domain alone, and publishes an accurate rendition of the full electronic text.

Implementation:

This strategy implemented using Python programming language. The Natural Language Toolkit (or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing (NLP) for English written in the Python programming language. ... NLTK supports classification, tokenization, stemming, tagging, parsing, and semantic reasoning functionalities. Scikit-Learn (sklearn library is used for building the models) and numpy library is used as a data structure. Matplotlib library is also used for visualizing the data, models' accuracy,etc.

Result & Analysis:

Preprocessing and cleaning:

We made a preprocessing and cleaning using functions to convert the words to lower case, remove punctuation, special characters and stopwords, stemming to remove the (-ing, -ly, ...) and Lemmatisation (for converting the word to the root word ,e.g.. plays-playing-played to play).

```
def preprocessing_cleaning(  book_text , flg_stemm=False, flg_lemm=True   ):
  book_text = book_text.lower()                                  # convert to lower case
  tokenizer = RegexpTokenizer(r'\w+')                                  # remove Punctuation and special characters
  tokens = tokenizer.tokenize(book_text)                              # Remove stop words
  book_words = [w for w in tokens if not w in stopwords.words('english')]
  # return " ".join(filtered_words)
                                              #               Stemming (remove -ing, -ly, ...)
  if flg_stemm == True:
      ps = nltk.stem.porter.PorterStemmer()
      book_words = [ps.stem(word) for word in book_words]

                                               #               Lemmatisation (convert the word into root word)
  if flg_lemm == True:
      lem = nltk.stem.wordnet.WordNetLemmatizer()
      book_words = [lem.lemmatize(word) for word in book_words]

  return book_words

# sentence = "At eight o'clock on Thursday morning Arthur didn't feel very good. French-Fries"
# print ( preprocessing_cleaning(sentence)  )
```

Partitioning:

We defined a function to get 200 random partitions of the books containing 150 words only and a loop to randomize and save the partition and the label.

```
[ ]
    def Partition_of_Books(  book_names , number_of_partitions  ,  number_of_words  ):
      book_partitions_list = [ ]    # list to save 200 partition for every book
      book_labels_list = [ ]        # list to save the label or book name for every partition
      for book in book_names:
       f = urlopen(book)
       text = f.read().decode('utf-8')
       book_words = preprocessing_cleaning( text  )
       # tokenized_word =  nltk.word_tokenize(text)      # to split our text to words
       # text = nltk.Text( tokenized_word )
       # text = [word for word in text if not word in stopwords.words()]

       for x in range( number_of_partitions ):
         randoms  = np.random.randint( 0 ,  (len( book_words ) - number_of_words ) )  # to randomize the location where we get the partition
         partition_words = ( book_words[ randoms : randoms + number_of_words ] )   # save the partition
         partition_string = " ".join( partition_words )
         book_partitions_list.append( partition_string )
         book_labels_list.append( book_names.index(book) )                                  # save the label
      return  book_partitions_list , book_labels_list
```

Feature engineering:

Bow and TF-IDF

We did it using Bow (bag of words) and TF-IDF. We calculated bag of words to get the frequency of every word in the text using CountVectorizer() . We also calculated the term-frequency (TF) and the inverse document frequency (idf) using TfidfTransformer().

```
[ ]  def Bow_Tf_Idf(   book_partitions_list  ):


       count_vect = CountVectorizer()
       X_train_counts = count_vect.fit_transform( book_partitions_list )  #
       tfidf_transformer = TfidfTransformer()
       X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)   # \
       # X_train_tfidf.shape
       return  X_train_tfidf
```

N-gram

We also used N-gram and calculated it as N-Grams model is one of the most widely used sentence-to-vector models since it captures the context between N-words in a sentence.

```
def ngram(book_partitions_list , n ):
  count_vect = CountVectorizer(ngram_range=(1,n))
  X_train_counts = count_vect.fit_transform( book_partitions_list )
  tfidf_transformer = TfidfTransformer()
  X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
  return  X_train_tfidf

c=ngram(X_train,3)
print(c.shape )
```

```
(700, 113672)
```

LDA (Latent Dirichlet Allocation)

So as for LDA, we calculated LDA as it is an example of topic model and it is used to classify text in a document to a particular topic. It builds a topic per document model and words per topic model, modeled as Dirichlet distributions. Here we are going to apply LDA to a set of documents and split them into topics.

```
[12] def lda(book_partitions_list) :
       count_vect = CountVectorizer()
       X_train_counts = count_vect.fit_transform( book_partitions_list )
       LDA = LatentDirichletAllocation( n_components=7,random_state=42 )
       LDA.fit(X_train_counts)
       return LDA.transform(X_train_counts)

     t=lda(book_partitions_list)
     t.shape
```

```
(1000, 7)
```

Visualizing the Data:

Word cloud for the all books and each one.

for all books book number

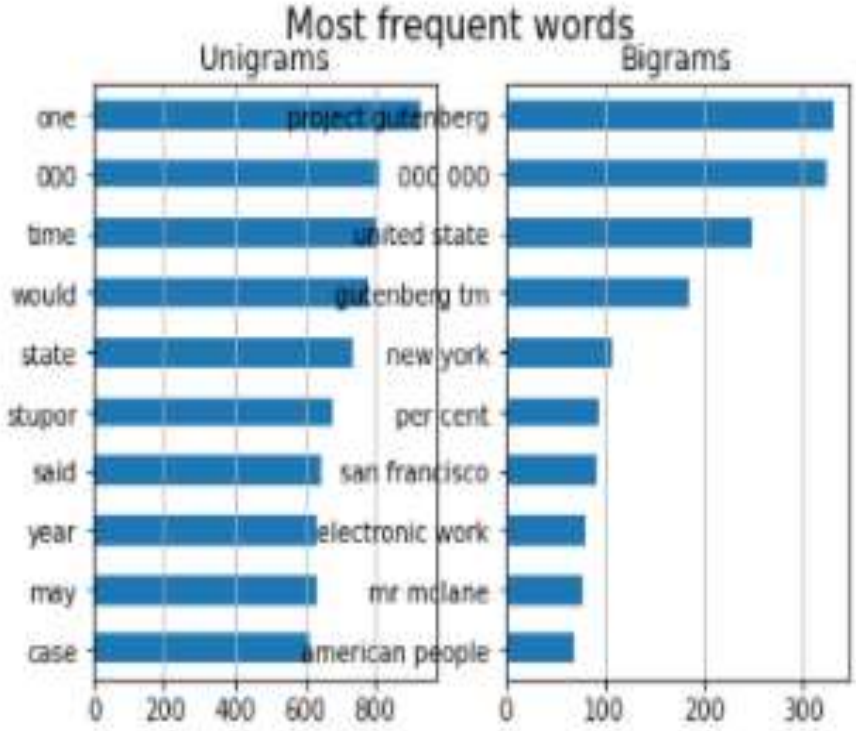
for book number 1

for book number 0

for book number 2

for book number 3

for book number 4

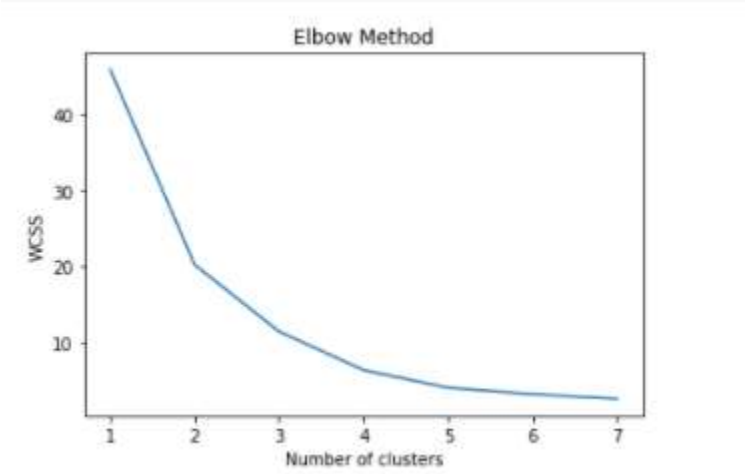


Building Models with the algorithms:

First, we tried different methods to find the number of clusters to be selected with WCSS and ElBow method

```
# X_tf_idf_feature = Bow_Tf_Idf( book_partitions_list )

wcss = []
for i in range(1, 8):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=0)
    kmeans.fit( Data_2D )
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 8), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

Based on the elbow curve our n_cluster will equal to 4.



We used PCA library to make the data with 2 dimensions in order to draw them

```
[ ]  pca = PCA(  n_components=2  )
     X_TF_IDF_features_2D = pca.fit_transform(  X_TF_IDF_features   )
     X_N_Gram_features_2d = pca.fit_transform(  X_N_Gram_features   )
     X_LDA_features_2d = pca.fit_transform(  X_LDA_features   )
```

Building K_means Model with TF_IDF , N_grams , LDA and word embedding:

```
[ ]  def Build_K_means( Data_2D ):

       kmeans = KMeans( n_clusters=5 , init='k-means++', max_iter=300, n_init=10, random_state=0)
       pred_y = kmeans.fit_predict(  Data_2D   )
       kappa=cohen_kappa_score(book_labels_list, pred_y )
       print(" kappa =",kappa)


       score = silhouette_score(X_LDA_features_2d, pred_y)
       print(" silhouette_score =",score)

       plt.scatter(  Data_2D[:,0],   Data_2D[:,1] )
       plt.scatter(  kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='red')
       plt.show(  )
```

Figure 1: Plotting K-means with Tf-idf

```
[ ]  Build_K_means( X_TF_IDF_features_2D )
```

```
kappa = -0.0024999999999999467
silhouette_score = 0.04064070025547258
```

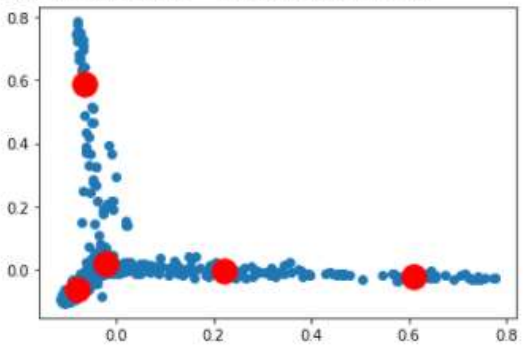


Figure 2: Plotting K-means with N-gram

```
[ ]  Build_K_means( X_N_Gram_features_2d )
```

```
kappa = -0.14625
silhouette_score = 0.06656317819473988
```

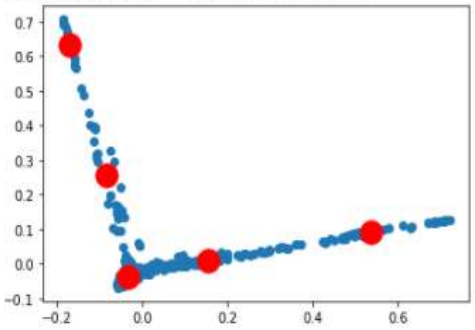


Figure 3: Plotting K-means with LDA

```
Build_K_means( X_LDA_features_2d )
```

```
kappa = -0.125
silhouette_score = 0.6098853989360109
```

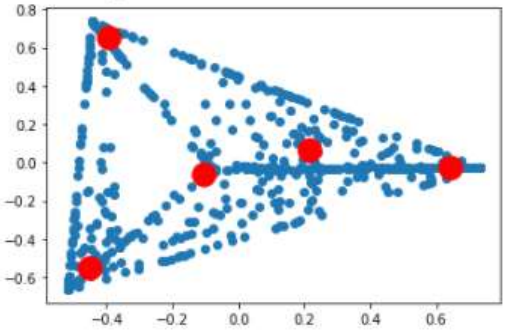
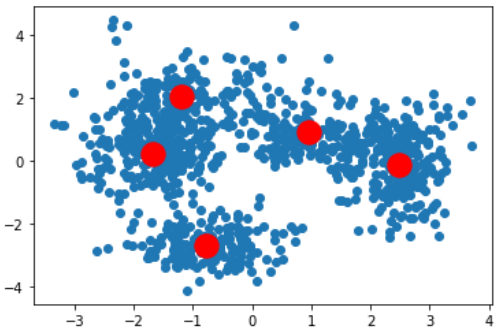
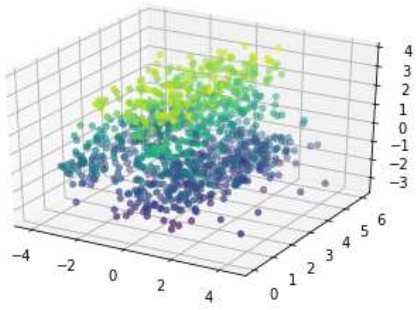


Figure 4: Plotting K-means with word embedding

```
kappa = -0.20750000000000002
silhouette_score = 0.20974507416897784
```





Building EM Model with TF_IDF , N_grams , LDA and word embedding:

Converting Data into DaraFrame

```
[23] def Convert_data_2DataFrame( X_train_feature ):
       data = pd.DataFrame(  X_train_feature )
       data.columns=["X1","X2"]
       data["cluster"]= book_labels_list
       # print( data.head(    ) )
       return data
```

```
[ ]  def Build_GaussianMixture( data ):

       gmm = GaussianMixture(  5 ,  covariance_type='full',  random_state=0).fit( data.iloc[ : , :2]   )
       labels = gmm.predict(   data.iloc[ : , :2]  )
       data[["predicted_cluster"]]=labels

       y_actual = data.iloc[ : , 2:3 ]
       y_predicted = labels
       # print( y_predicted )

       kappa2 = cohen_kappa_score(y_actual , y_predicted  )

       print( " kappa = ",kappa2  )

       score = silhouette_score(X_LDA_features_2d, y_predicted)
       print(" silhouette_score =",score)

       plt.figure(figsize=(9,7))
       sns.scatterplot(data=data,
                       x="X1",
                       y="X2",
                       hue="predicted_cluster",
                       palette=[ "Black" , "red","blue","green", "purple"])
       plt.savefig("fitting_Gaussian_Mixture_Models_with_3_components_scikit_learn_Python.png",
                           format='png',dpi=150)
```

Plotting actual data with tf_idf:

```
[ ] data = Convert_data_2DataFrame(  X_TF_IDF_features_2D  )
    plt.figure(figsize=(9,7))
    sns.scatterplot(data=data,
                    x="X1",
                    y="X2",
                    hue="cluster",
                    palette=[ "Black" , "red","blue","green", "purple"])
    plt.savefig("Data_for_fitting_Gaussian_Mixture_Models_Python.png",
                        format='png',dpi=150)
```

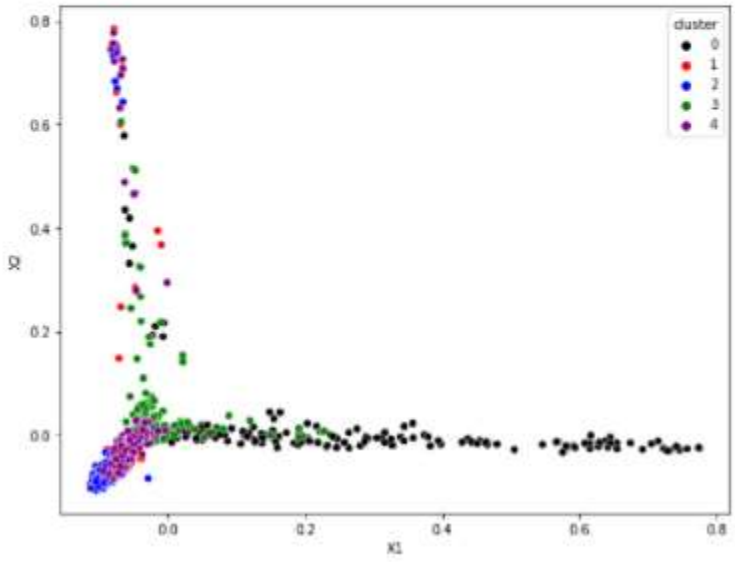


Figure 1: Plotting EM with Tf_idf

```
] data = Convert_data_2DataFrame(  X_TF_IDF_features_2D  )
  Build_GaussianMixture( data )
```

```
kappa =  -0.06624999999999992
silhouette_score = 0.04543412021706834
```

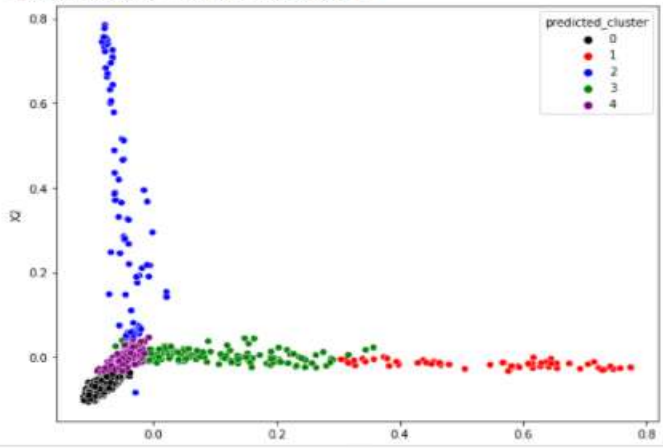


Figure 2: Plotting EM with N-gram

```
[ ] data = Convert_data_2DataFrame(  X_N_Gram_features_2d  )
    Build_GaussianMixture( data )
```

```
kappa =  -0.23249999999999993
silhouette_score = 0.029685470849319117
```

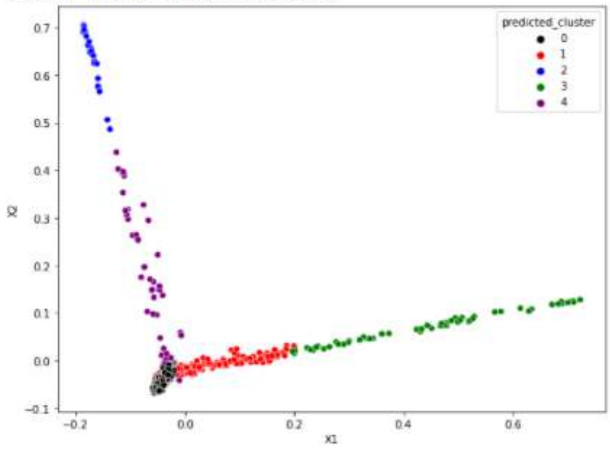


Figure 3: Plotting EM with LDA

```
[ ]  data = Convert_data_2DataFrame(  X_LDA_features_2d  )
     Build_GaussianMixture( data )
```

```
kappa =  0.06499999999999995
silhouette_score = 0.29676058171803243
```

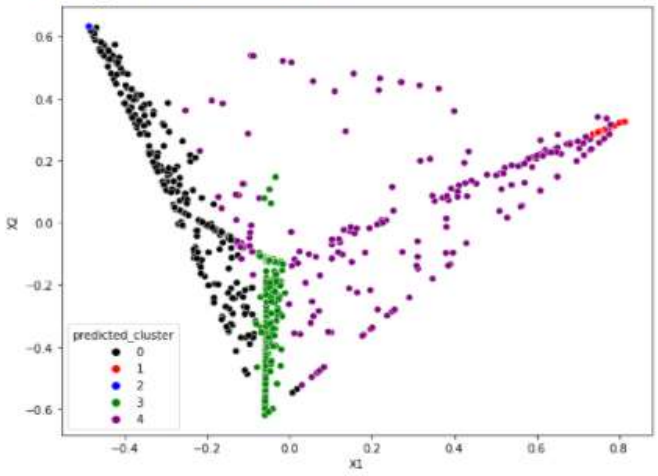
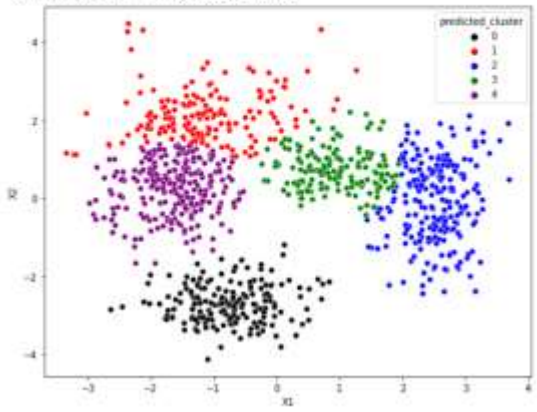


Figure 4: Plotting EM with Word embedding

```
data = Convert_data_2DataFrame(  X_Word2Vec_feature_2d  )
Build_GaussianMixture( data )
```

```
kappa =  -0.11250000000000004
silhouette_score = 0.18290995313017056
```



Building Hierarchical Model with TF_IDF , N_grams , LDA and word embedding:

```
def Build_Hierarchical_Model( z ):

  plt.figure(figsize=(10, 7))
  plt.title(" Books' partition Dendograms")
  dend = shc.dendrogram(shc.linkage(z, method='ward'))

  cluster = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')
  cluster.fit_predict(z)
  # print( cluster.labels_ )
  plt.figure(figsize=(9, 5))
  plt.scatter(z[:,0], z[:,1], c=cluster.labels_ , cmap='rainbow' )
  y_actual = data.iloc[ : , 2:3 ]
  kappa2 = cohen_kappa_score(y_actual , cluster.labels_  )

  print( " kappa = ",kappa2  )

  score = silhouette_score(X_LDA_features_2d, cluster.labels_ )
  print(" silhouette_score =",score)
```

Figure 1: Plotting Hierarchical with Tf_idf

```
Build_Hierarchical_Model(X_TF_IDF_features_2D)
```

```
kappa -0.15749999999999997
 silhouette_score = 0.02461954540998014
```

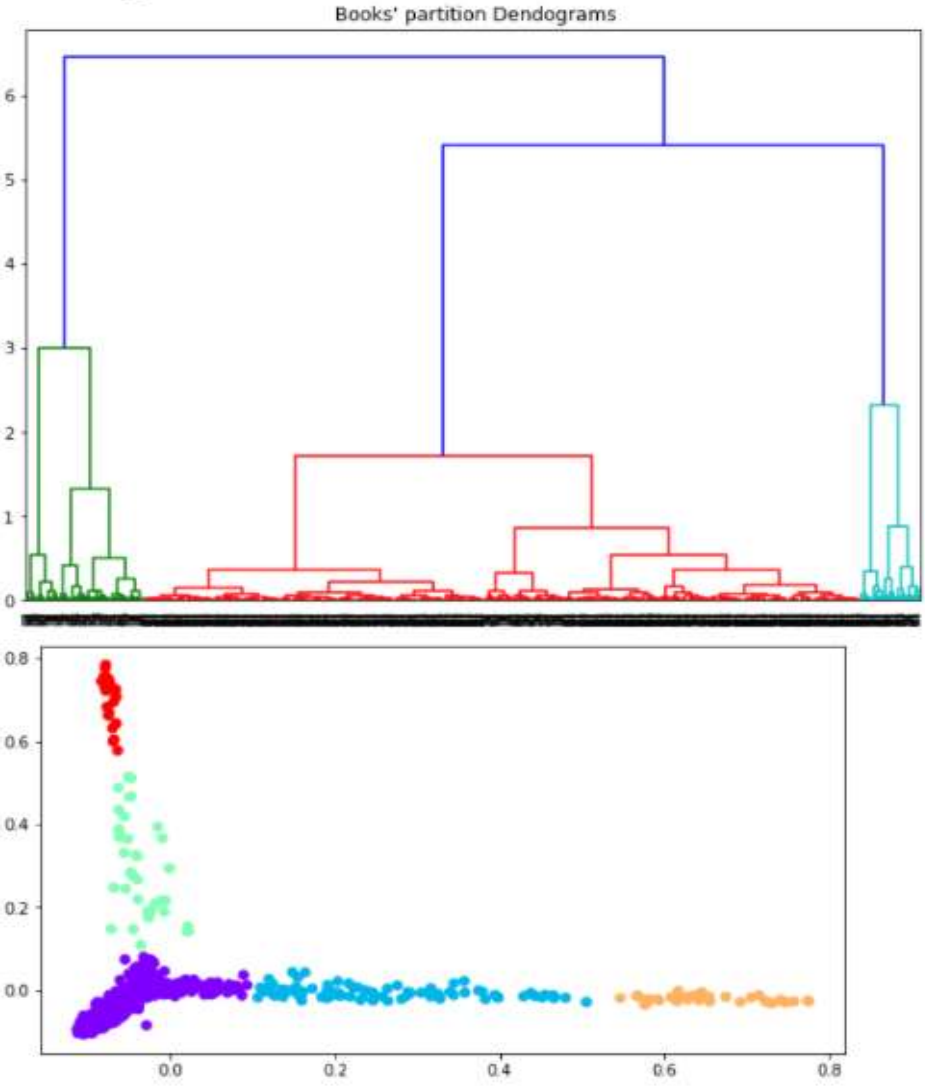
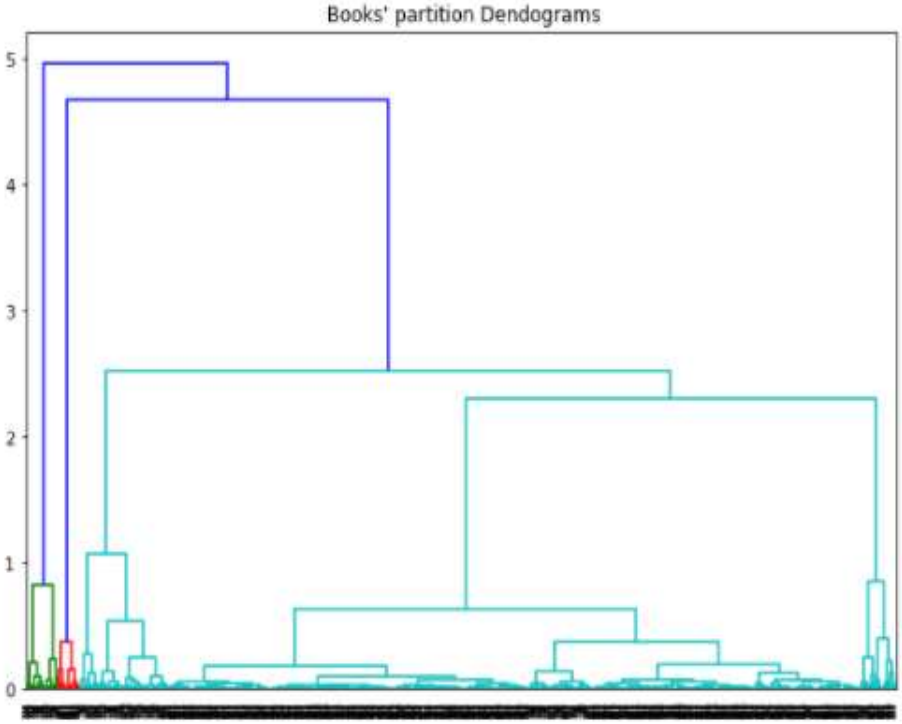


Figure 2: Plotting Hierarchical with N-gram

```
Build_Hierarchical_Model(X_N_Gram_features_2d)
```

```
kappa 0.11375000000000002
 silhouette_score = 0.03237118382278515
```



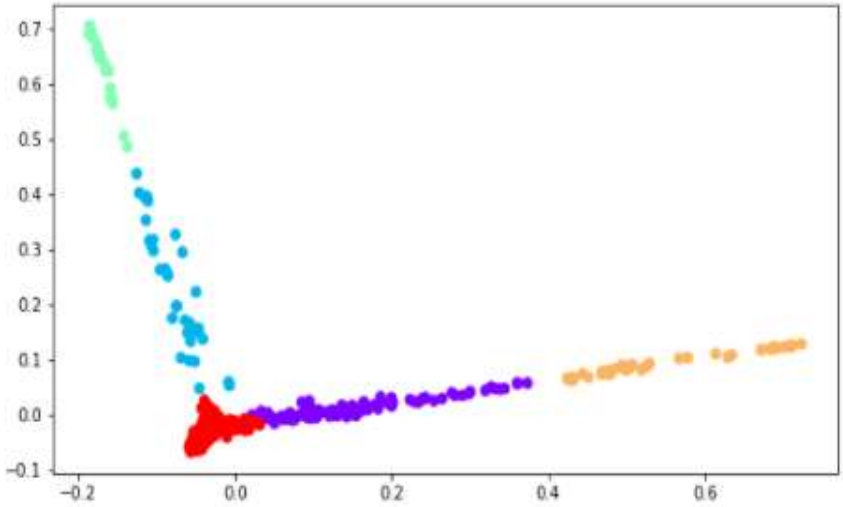


Figure 3: Plotting Hierarchical with LDA

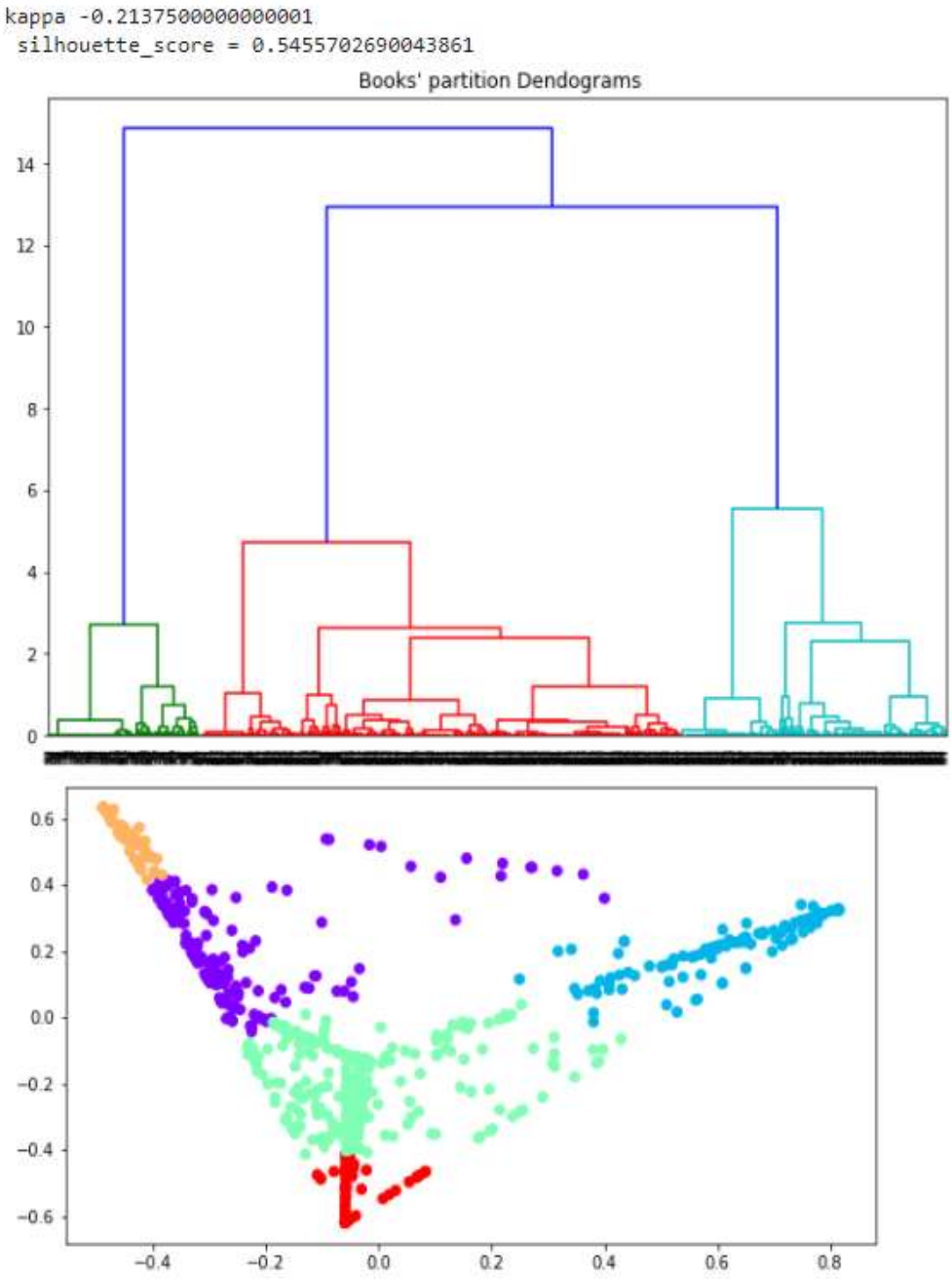
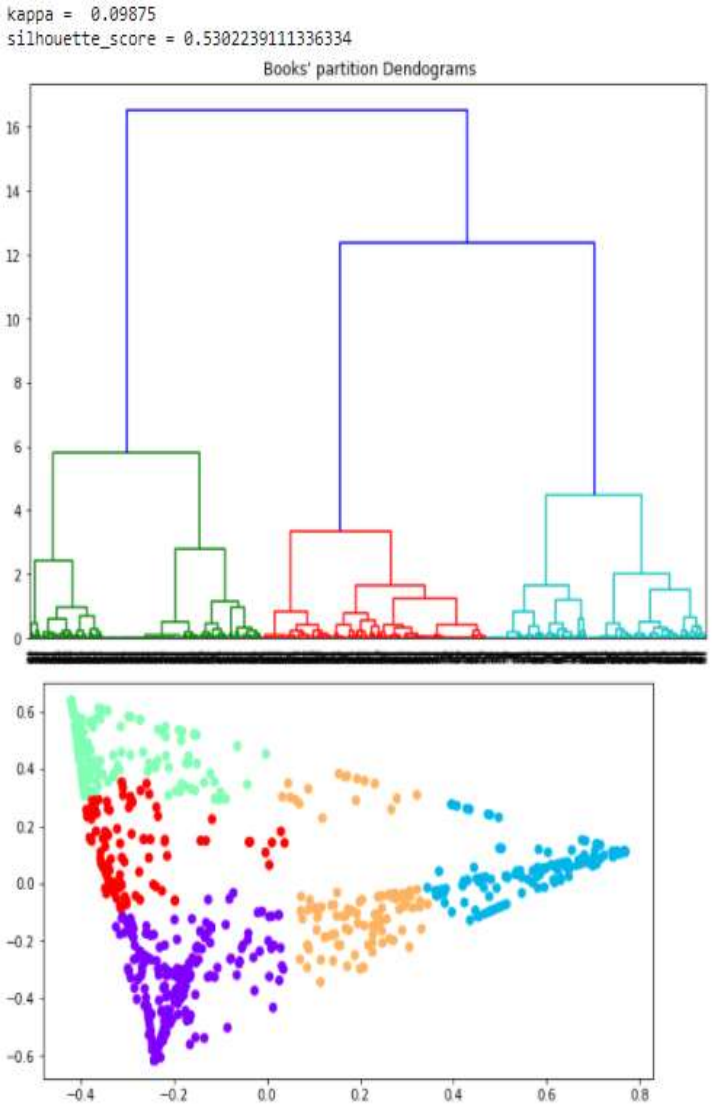


Figure 4: Plotting Hierarchical with Word embedding



For n_cluster = 4

Building K_means Model with TF_IDF , N_grams , LDA and word embedding:

Figure 1: Plotting K-means with Tf-idf

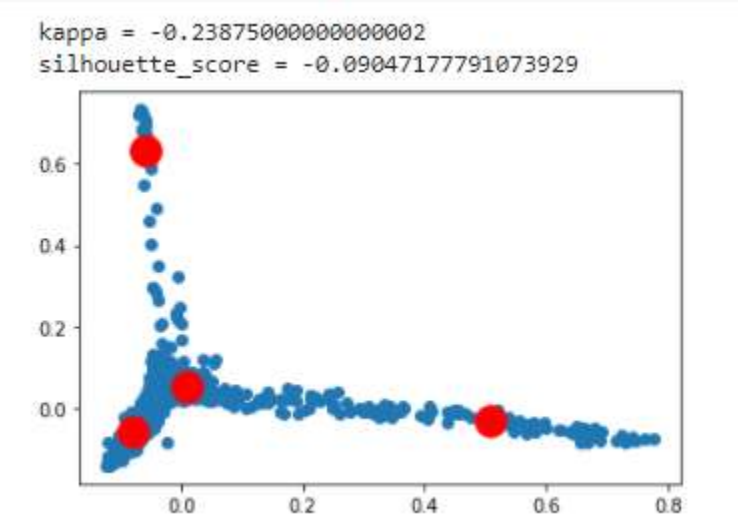


Figure 2: Plotting K-means with N-gram

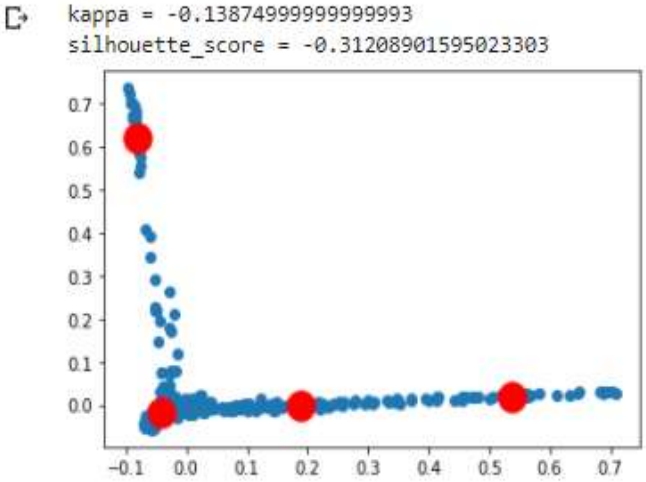


Figure 3: Plotting K-means with LDA

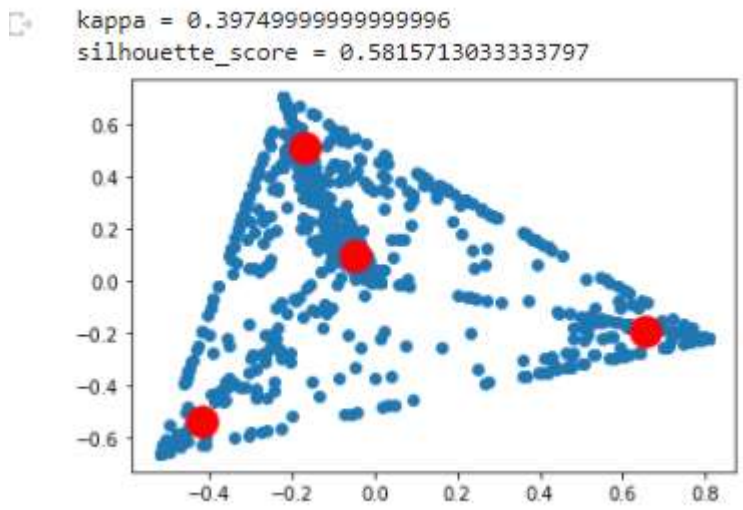
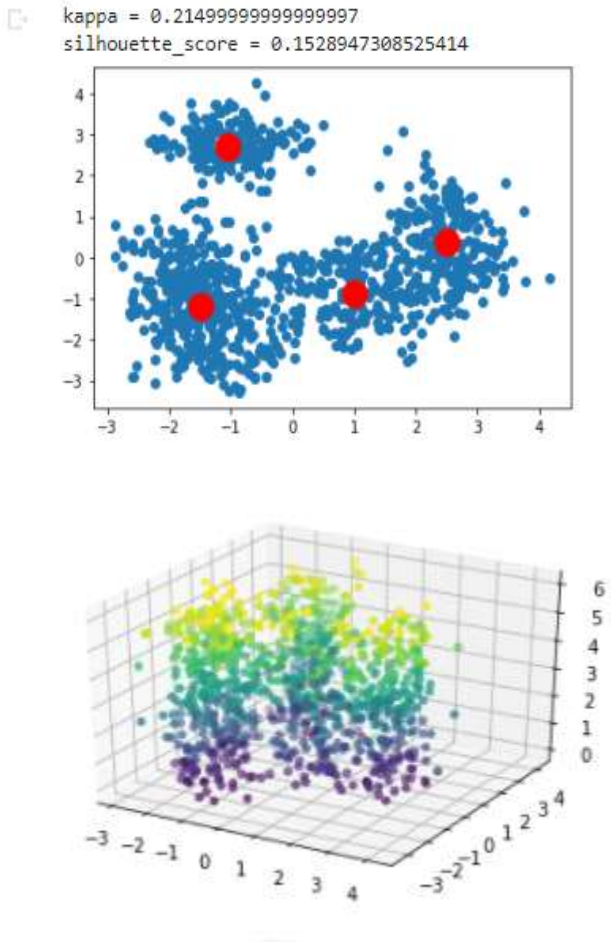


Figure 4: Plotting K-means with word embedding



Building EM Model with TF_IDF , N_grams , LDA and word embedding:

Converting Data into DaraFrame

```
[23] def Convert_data_2DataFrame( X_train_feature ):
       data = pd.DataFrame(  X_train_feature )
       data.columns=["X1","X2"]
       data["cluster"]= book_labels_list
       # print( data.head(    ) )
       return data
```

-

```
[ ]  def Build_GaussianMixture( data ):

     gmm = GaussianMixture(  5 ,  covariance_type='full',  random_state=0).fit( data.iloc[ : , :2]   )
     labels = gmm.predict(   data.iloc[ : , :2]  )
     data[["predicted_cluster"]]=labels

     y_actual = data.iloc[ : , 2:3 ]
     y_predicted = labels
     # print( y_predicted )

     kappa2 = cohen_kappa_score(y_actual , y_predicted  )

     print( " kappa = ",kappa2  )

     score = silhouette_score(X_LDA_features_2d, y_predicted)
     print(" silhouette_score =",score)

     plt.figure(figsize=(9,7))
     sns.scatterplot(data=data,
                     x="X1",
                     y="X2",
                     hue="predicted_cluster",
                     palette=[ "Black" , "red","blue","green", "purple"])
     plt.savefig("fitting_Gaussian_Mixture_Models_with_3_components_scikit_learn_Python.png",
                       format='png',dpi=150)
```

Figure 1: Plotting EM with Tf_idf

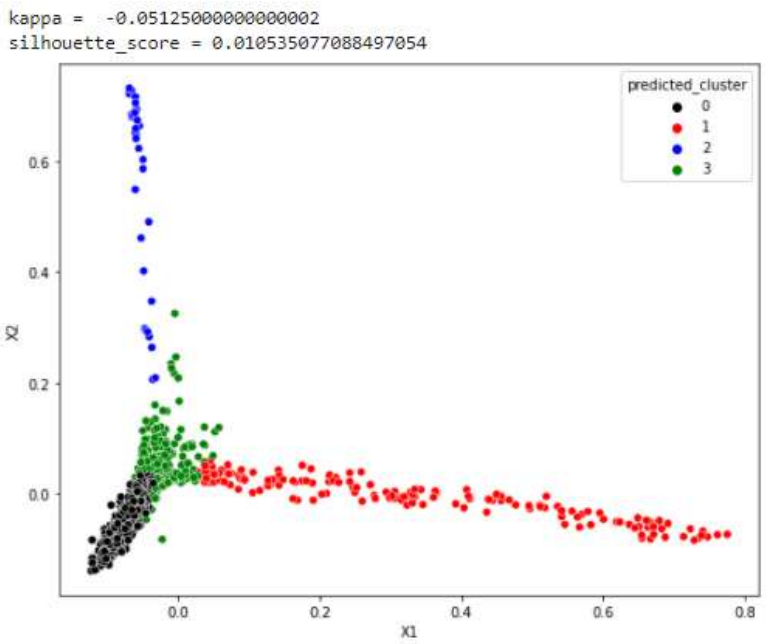


Figure 2: Plotting EM with N-gram

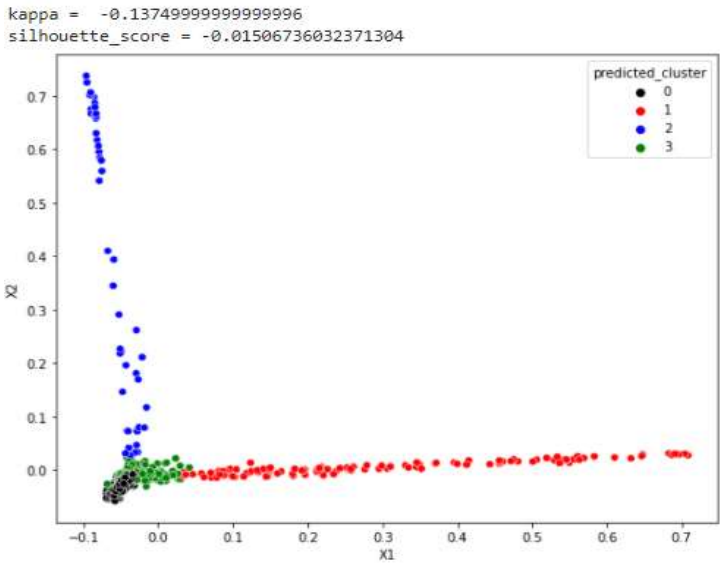


Figure 3: Plotting EM with LDA

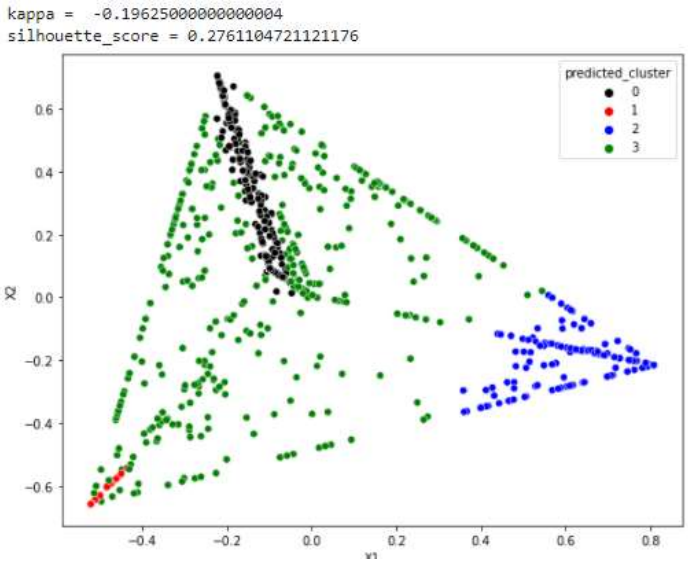
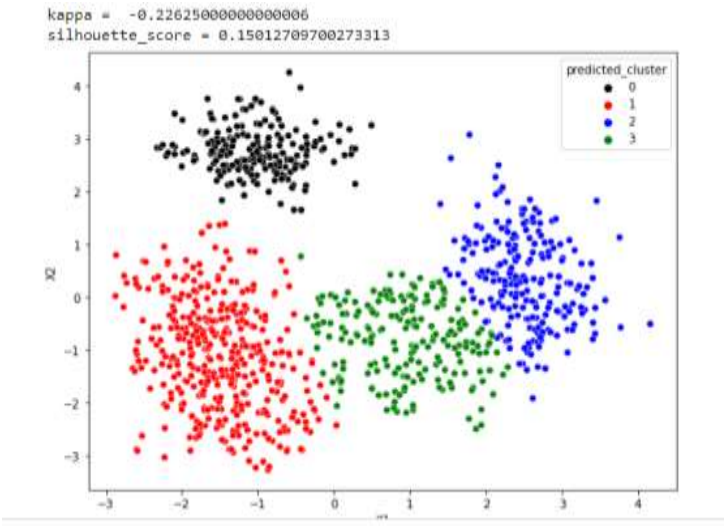


Figure 4: Plotting EM with Word embedding



Building Hierarchical Model with TF_IDF , N_grams , LDA and word embedding:

```
def Build_Hierarchical_Model( z ):

  plt.figure(figsize=(10, 7))
  plt.title(" Books' partition Dendograms")
  dend = shc.dendrogram(shc.linkage(z, method='ward'))

  cluster = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')
  cluster.fit_predict(z)
  # print( cluster.labels_ )
  plt.figure(figsize=(9, 5))
  plt.scatter(z[:,0], z[:,1], c=cluster.labels_ , cmap='rainbow' )
  y_actual = data.iloc[ : , 2:3 ]
  kappa2 = cohen_kappa_score(y_actual , cluster.labels_  )

  print( " kappa = ",kappa2  )

  score = silhouette_score(X_LDA_features_2d, cluster.labels_ )
  print(" silhouette_score =",score)
```

Figure 1: Plotting Hierarchical with Tf_idf

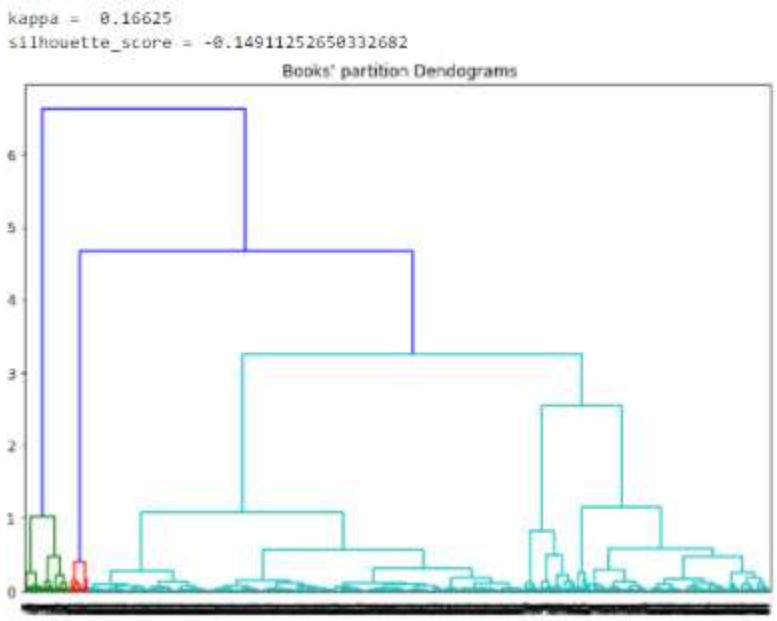


Figure 2: Plotting Hierarchical with N-gram

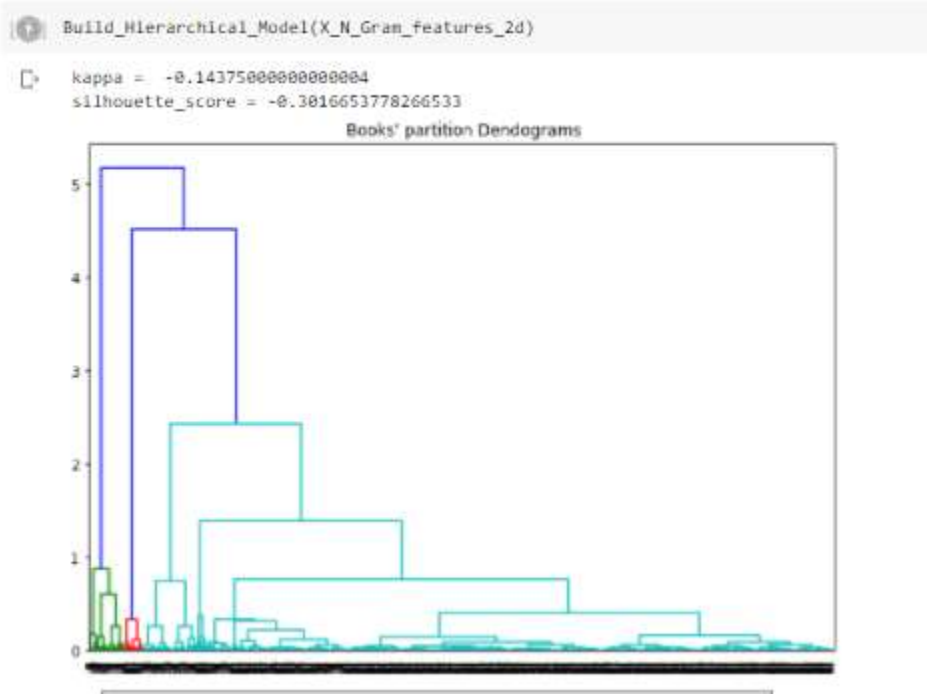


Figure 3: Plotting Hierarchical with LDA

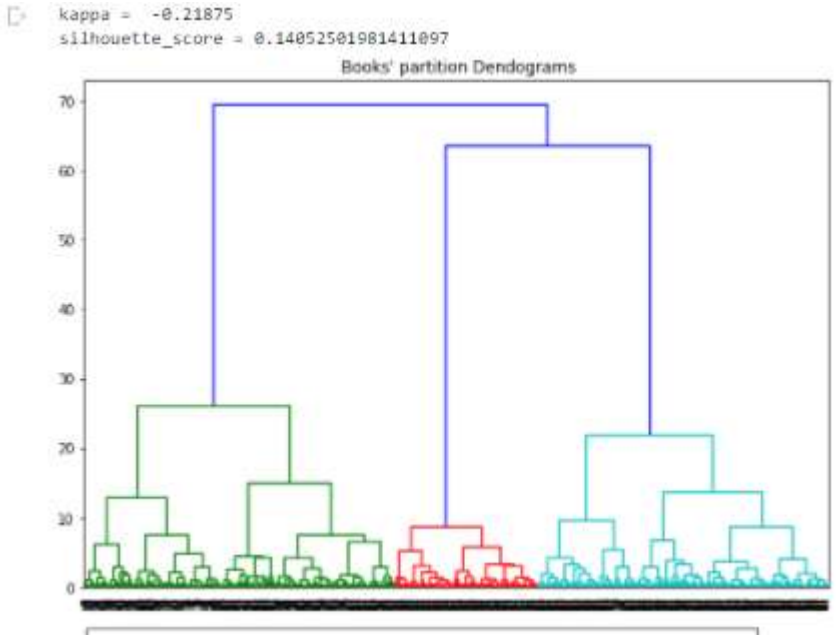
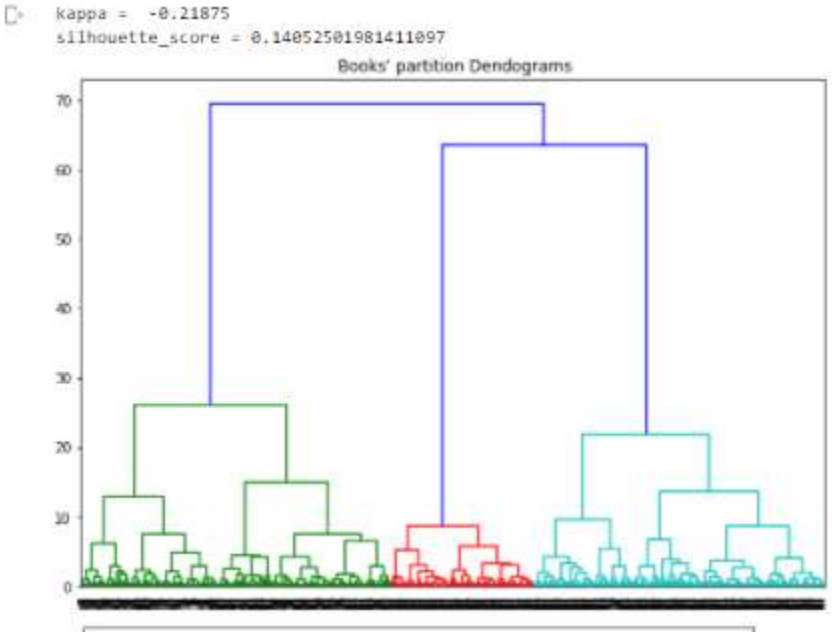


Figure 4: Plotting Hierarchical with Word embedding



Evaluation models:

For n_cluster =5

Table-1, shows Kappa of the models.

```
kappa=cohen_kappa_score(book_labels_list, pred_y )
print(" kappa =",kappa)
```

Table-2, shows Silhouette of the models

```
score = silhouette_score(X_LDA_features_2d, pred_y)
print(" silhouette_score =",score)
```

Kappa evaluation

Model	BOW+Tf_idf	N-gram	LDA	Word embedding
K-means	-0.002499	-0.14625	-0.08749	-0.20750
EM	-0.06624	-0.23249	0.06499	-0.11250
Hierarchical	-0.157499	0.113750	-0.21375	0.09875

Silhouette evaluation

Model	BOW+Tf_idf	N-gram	LDA	Word embedding
K-means	0.04064	0.06656	0.55708	0.020975
Em	0.04543	0.029685	0.29676	.182909
Hierarchical	0.02461	0.032371	0.54557	0.53022

For n_cluster =4

Table-1, shows Kappa of the models.

```
kappa=cohen_kappa_score(book_labels_list, pred_y )
print(" kappa =",kappa)
```

Table-2, shows Silhouette of the models

```
score = silhouette_score(X_LDA_features_2d, pred_y)
print(" silhouette_score =",score)
```

Kappa evaluation

Model	BOW+Tf_idf	N-gram	LDA	Word embedding
K-means	-0.23875	-0.13874	0.3974	0.2149
EM	-0.05125	-0.13749	-0.19625	-0.22625
Hierarchical	0.16625	-0.14375	0.19875	-0.21875

Silhouette evaluation

Model	BOW+Tf_idf	N-gram	LDA	Word embedding
K-means	-0.09047	-0.31208	0.5815	0.1528
Em	0.01054	-0.01506	0.27611	0.15013
Hierarchical	-0.14911	-0.30167	0.57366	0.14053

For the coherence:

Doing the coherence with lda model with Umass and C_V

```
def calc_coh(links):
  texts=[]
  for link in links:
    f = urlopen(link)
    myfile = f.read().decode('utf-8')
    t = preprocessing_cleaning(myfile)
    texts.append(t)
  dictionary = corpora.Dictionary(texts)
  corpus = [dictionary.doc2bow(text) for text in texts]
  tfidf = models.TfidfModel(corpus)
  corpus_tfidf = tfidf[corpus]
  lda = models.LdaModel(corpus_tfidf, id2word=dictionary, num_topics=20, passes=3)
  cm = CoherenceModel(model=lda, corpus=corpus, coherence='u_mass')
  cm2 = CoherenceModel(model=lda, texts=texts, coherence='c_v')
  coherence = cm.get_coherence()  # get coherence value
  coherence_v = cm2.get_coherence()

  print("coherence with u_mass ",coherence ,"coherence with c_v " , coherence_v)


  return corpus,lda,dictionary
```

Output :

The first one with u_mass = - 0.06994586

The second one with C_v = 0.64530739

```
[ ] book_names = [ 'https://www.gutenberg.org/cache/epub/7361/pg7361.txt', 'https://www.gutenberg.org/cache/epub/6884/pg6884.txt', 'https://www.gutenbe
    corpus,lda,dictionary=calc_coh( book_names)

    /usr/local/lib/python3.7/dist-packages/gensim/models/ldamodel.py:807: RuntimeWarning: overflow encountered in exp2
      perwordbound, np.exp2(-perwordbound), len(chunk), corpus_words
    coherence with u_mass  -0.6994586544721623 coherence with c_v  0.645307392925036
```

And after filtering ,the coherence is as shown below:

```
[ ]  topics = [

               [ 'disease', 'hospital', 'mind'] ,
             ]

     cm3 = CoherenceModel(topics=topics, corpus=corpus, dictionary=dictionary, coherence='u_mass')
     coherence_after_filter = cm3.get_coherence()

     print(coherence_after_filter)
```

```
1.24988908112222e-12
```

Error analysis:

For n_cluster = 5

According to the results after computing the Silhouette and Kappa. We found that the highest kappa and Silhouette is the Hierarchical with N-gram features. So, we consider the Hierarchical as the champion model to do our error analysis on.

For n_cluster = 4

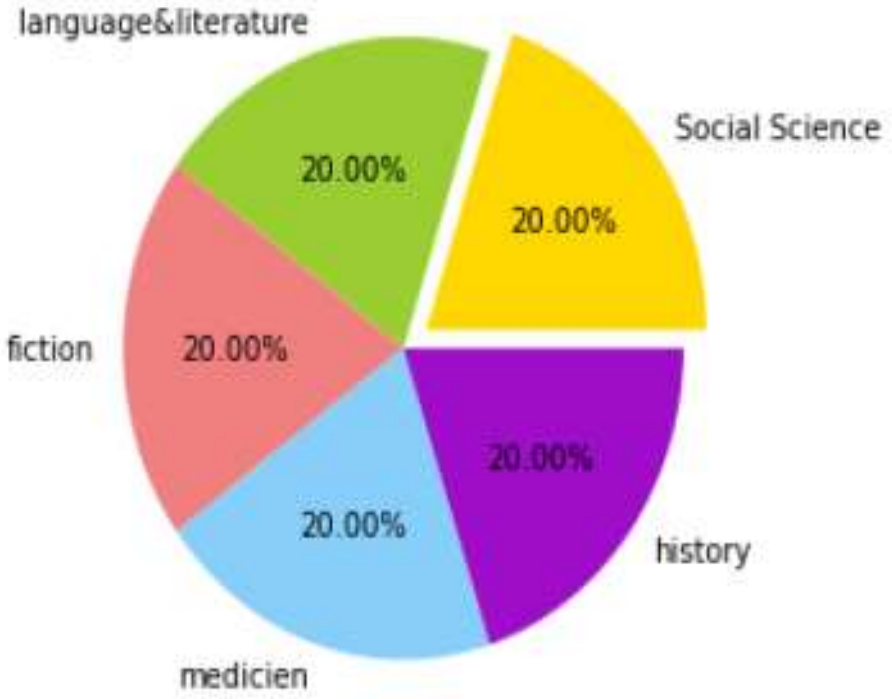
According to the results after computing the Silhouette and Kappa. We found that the highest kappa and Silhouette is the K-means with LDA. So, we consider the K-means as the champion model to do our error analysis on.

We defined a function to detect the mis-classified cases then we identified the correct labels for these records.

```
[ ]  error_partition =  [ ]
     error_books = [ ]
     books = [ "social science " ,"language& literatures","fiction","History","medicien"]
     def detect_misclassified( actual , predicted  ):
       count0,count1,count2,count3,count4=0,0,0,0,0
       count5,count6,count7,count8,count9=0,0,0,0,0


       #print( len( actual ) )
       #print( len( predicted ) )
       error_partition = [ ]

       for x in range( len( actual ))  :
         #print(actual[x] ,  predicted[x])
         if(   actual[x] !=  predicted[x] ):
           #print( "predict as  = " ,  books[  predicted[x]  ] ,"actual = " , books[ actual[x] ]   )
           if books[  predicted[x] ] == books[ 0]:
               count0=count0+1
           elif books[ predicted[x] ] == books[ 1]:
               count1=count1+1
           elif books[predicted[x]  ] == books[ 2]:
               count2=count2+1
           elif books[ predicted[x]  ] == books[3]:
               count3=count3+1
           else:
               count4=count4+1
           error_books.append(  book_names[ int( predicted[x] ) ]  )
           error_partition.append( book_partitions_list[x] )
```

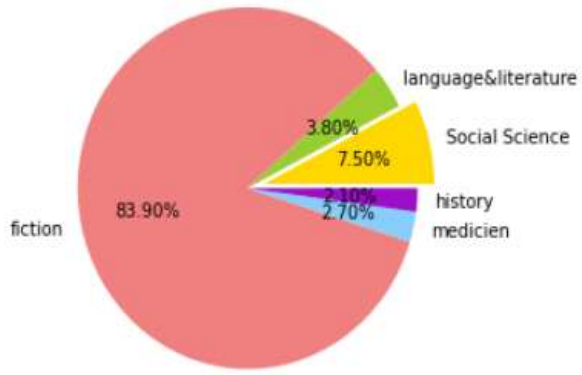
```
else:
  #print( "predict as  = " ,  books[  predicted[x]  ] ,"actual = " , books[ actual[x] ]   )
  if(   actual[x] ==  predicted[x] ):
  #print( "predict as  = " ,  books[  predicted[x]  ] ,"actual = " , books[ actual[x] ]   )
    if books[  predicted[x] ] == books[ 0]:
        count5=count5+1
    elif books[ predicted[x] ] == books[ 1]:
        count6=count6+1
    elif books[predicted[x]  ] == books[ 2]:
        count7=count7+1
    elif books[ predicted[x]  ] == books[3]:
        count8=count8+1
    else:
        count9=count9+1
```

```
    # Plot
    plt.pie(sizes, explode=explode, labels=labels, colors=colors,autopct='%1.2f%%')

    plt.axis('equal')
    plt.show()

    count0,count1,count2,count3,count4=0,0,0,0,0
    for x in range(len(y_pred)) :
      if books[  y_pred[x] ] == books[ 0]:
               count0=count0+1
      elif books[ y_pred[x] ] == books[ 1]:
          count1=count1+1
      elif books[y_pred[x]  ] == books[ 2]:
          count2=count2+1
      elif books[ y_pred[x]  ] == books[3]:
          count3=count3+1
      else:
        count4=count4+1
```

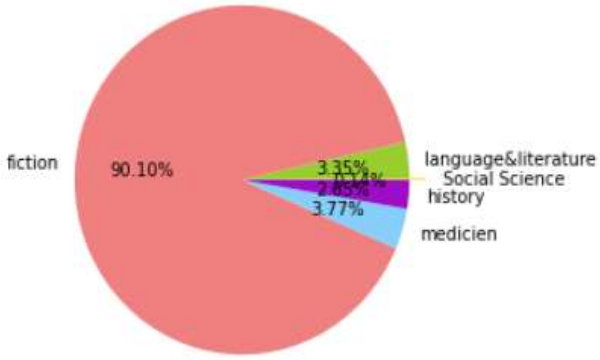


This figure are the actual ones that should be predicted

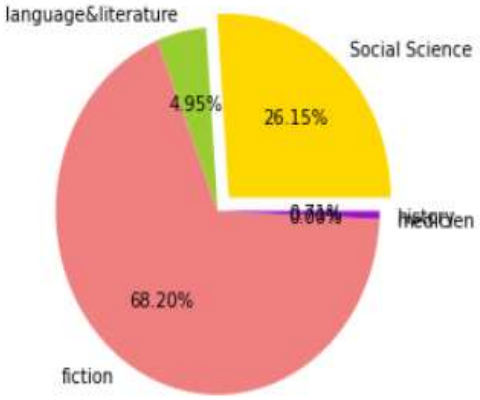
These figures about what is predicted and the error and corrected ones .



model predict social science 75
model predict s language& literature 38
model predict fiction 839
model predict History 27
model predict medicien 21



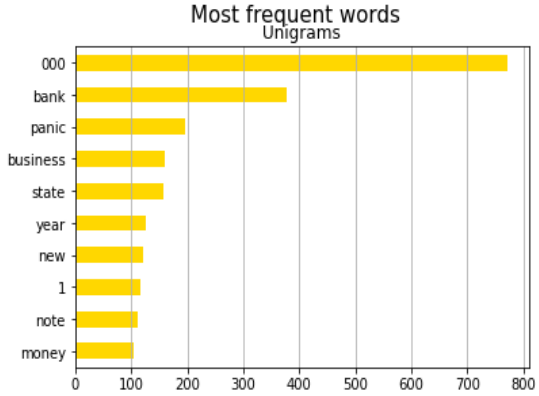
model predict social science Error 1
model predict s language& literature Error 24
model predict fiction Error 646
model predict History Error 27
model predict medicien Error 19
Error Partitions Number is 717



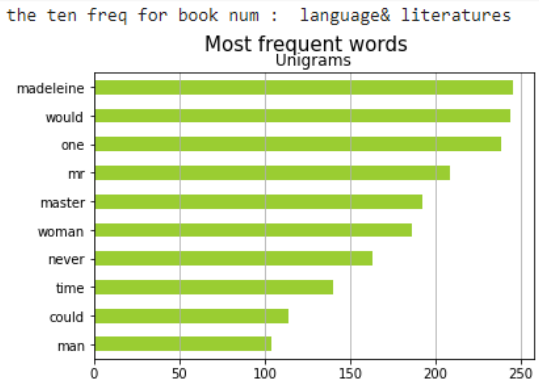
model predict social science correct 74
model predict in language& literature correct 14
model predict fiction correct 193
model predict in History correct 0
model predict in medicien correct 2
correct Partitions Number is 283

These figures show the ten frequent words for the books to know why it confused the machine and there was bias to the fiction one.

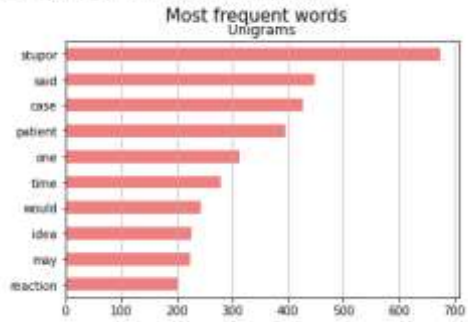
the ten freq for book num : social science



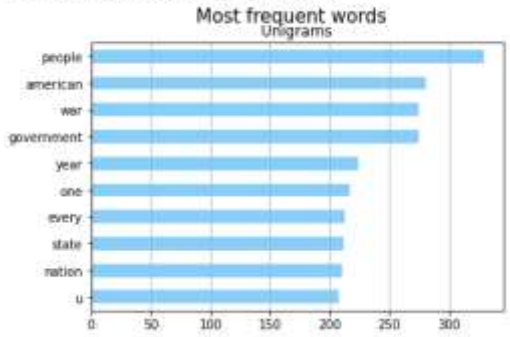
the ten freq for book num : language& literatures



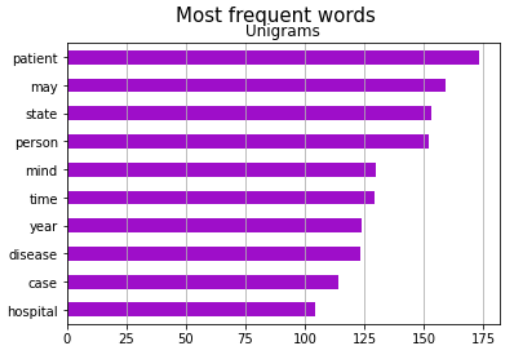
the ten freq for book num : fiction



the ten freq for book num : History

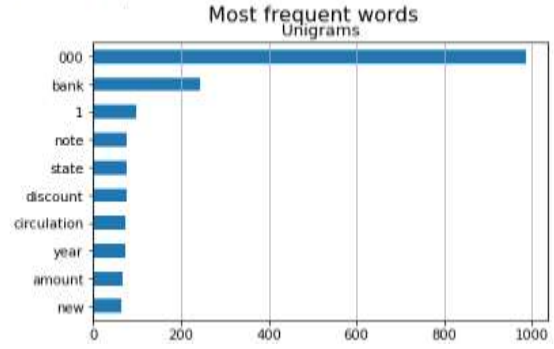


the ten freq for book num : medicien

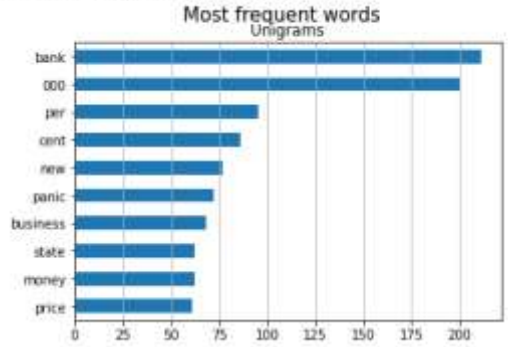


For each cluster

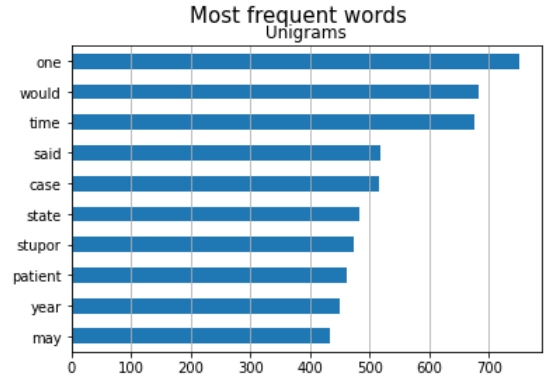
the ten freq for cluster num : 0



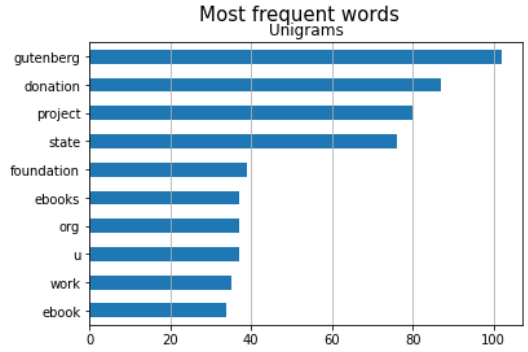
the ten freq for cluster num : 1



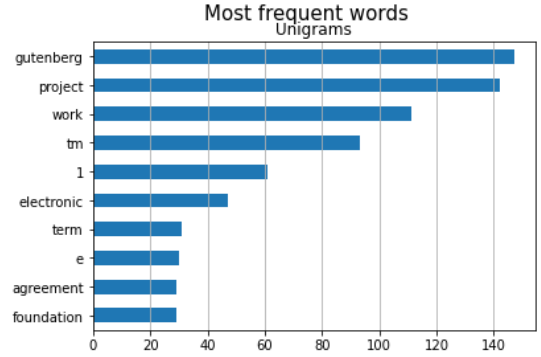
the ten freq for cluster num : 2



the ten freq for cluster num : 3



the ten freq for cluster num : 4



Conclusion:

We implemented the steps (preparing till verifying and validating) using by using the K-means, EM. In addition to, implementing the Hierarchical clustering algorithm for text Clustering. We found that Hierarchical with n-gram provides the best performance among them from Kappa and Silhouette perspective.