University of Ottawa
Faculty of Engineering

School of Electrical Engineering
and Computer Science

uOttawa

2021 Summer DTI5125 Data Science Applications

Assignment Three

Classification Assignment (Group) Report

Group Members:
Salma Mousa, Tokka Hassan, Reyad Melies, Shahenda Youssef

Group ID.: DSA_202101_ 10

Under Supervision: Dr. Arya Rahgozar

Submission Date: 20/06/2021

Table of Contents

LIST OF FIGURES

INTRODUCTION

In some real life application, data is not always labeled and the process can be both time and financially exhausting, and hence using clustering techniques can be very important in many business cases. Since clustering is an unsupervised learning technique, we might have to accept clusters that don't really represent our data well, so it's very important to investigate how the different clustering algorithms work, and study the results they output, so that we can make sure we are tuning our models correctly and according to our application.

For our project, we are using clustering algorithms on different books and checking what errors cause the model to give us inaccurate results, this can give very important insights for projects that contain large number of document analysis.

1 REPORT BODY

1.1 Data Preparation

1.1.1 Books Choice

For this project, we choose 5 books that are semantically different, written by different authors and have different genres.

The books share something in common that we believe will cause some issues with the clustering algorithms, they all revolve in different ways about biblical analogies and writings.

The books are:

1. Bible:
   Genre: religious text
2. Father brown by Chesterton:
   Genre: Mystery and fiction
   Points of interest: The book is about the adventures of a priest solving crimes
3. Macbeth by Shakespeare:
   Genre: tragedy
   Points of interest: The book has many biblical parallels with the bible and the old testament stories
4. Paradise Lost by John Milton:
   Genre: Epic poetry, Christian mythology
   Points if interest: The poems all revolve around biblical stories
5. Moby dick by Herman Melville:
   Genre: Novel, Adventure fiction
   Points of interest: the main character is inspired from the bible and there are biblical analogies.

1.1.2 Clean Data

The first step in data preparation is data cleaning from any unwanted data that can affect the model results.

We removed all non-alphabetic characters, stop words and then we used the stemming function to reduce the word to its stem.

```
text = re.sub(r'[^\w\s]', '', str(text).lower().strip())
text = re.sub(r'[\d_]', '', text)

print("--- removing stop words ---")          #stop words
word_tokens = word_tokenize(text)
filtered_words = []
for w in word_tokens:
    if w not in stop_words:
        filtered_words.append(w)

print("--- stemming ---")                       #stemming
new_text = stemming_text(filtered_words)
```

Figure 1 – Removing non-alphabetic characters code

Now the data consists of only words in lower case separated by white spaces and reduced to its stem.

1.1.3 Partition Data

After cleaning the data, it is ready for partitioning. We started by splitting the text to words, then we made a list of partitions each of 150 words. Finally, we took random 200 partitions and repeated the steps for five books of the same genre.

All partitions and their book labels are added to a pandas dataframe.

```
for word in filtered_sentence:
  temp.append(word)
  if len(temp) == 150:
   list_of_lists.append(temp)
   temp=[]

for i in range(200):
 ran = random.randint(0, len(list_of_lists)-1)
 list_of_lists1.append([listToString(list_of_lists[ran]), x])

dataFrame = pd.DataFrame(list_of_lists1, columns=["partition", "book"])
global dataFrameT
dataFrameT = dataFrameT.append(dataFrame, ignore_index = True)
```

Figure 2 - Text partitioning code

And the resulted dataframe is as follows.

	partition	book
0	fight battl peopl rest upon word hezekiah king...	bible-kjv.txt
1	went wife conceiv bare son call name beriah we...	bible-kjv.txt
2	god brought thee land egypt open thi mouth wid...	bible-kjv.txt
3	gold cover purpl midst thereof pave love daugh...	bible-kjv.txt
4	retir battl benjamin began smite kill men isra...	bible-kjv.txt

Figure 3 - Dataframe after text partitioning

1.1.4 Label Data

Now the dataframe is ready with the partitions and their book names, we can start labeling the data. This is simply done by separating the authors name from the book label, then adding it to the dataframe as a new column.

	partition	Author
0	fight battl peopl rest upon word hezekiah king...	bible
1	went wife conceiv bare son call name beriah we...	bible
2	god brought thee land egypt open thi mouth wid...	bible
3	gold cover purpl midst thereof pave love daugh...	bible
4	retir battl benjamin began smite kill men isra...	bible

Figure 4 - Dataframe after labeling partitions

1.2 Feature Extraction

1.2.1 Bag of Words

For implementing the bag of words method, we used the count vectorizer approach, where the resultant represents a sparse matrix of words that are most frequent in the partitions, with a maximum number of 2500 words.

```python
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer(max_features = 2500)
X_train_BoW = count_vect.fit_transform(dataFrameT["partition"])
X_train_BoW
```

```
<600x2500 sparse matrix of type '<class 'numpy.int64'>'
	with 42615 stored elements in Compressed Sparse Row format>
```

Figure 5 – bag of words code

	aaron	abhor	abid	abject	abl	abner	abomin	abraham	a
0	0	0	1	0	3	0	0	0	
1	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	

Figure 6 – dataset obtained using BoW method

1.2.2 TFIDF

The second test we prepared is using the TFIDF vectorizer which transforms text to feature vectors that shows the importance of each word using the inverse document frequency calculation, this this data can be used as input to estimator. We set the parameter "max-features" to 2500.

	aaron	abhor	abid	abject	abl	abner	abomin	abraham
0	0.0	0.0	0.062441	0.0	0.174892	0.0	0.0	0.0
1	0.0	0.0	0.000000	0.0	0.000000	0.0	0.0	0.0
2	0.0	0.0	0.000000	0.0	0.000000	0.0	0.0	0.0
3	0.0	0.0	0.000000	0.0	0.000000	0.0	0.0	0.0
4	0.0	0.0	0.000000	0.0	0.000000	0.0	0.0	0.0

Figure 7 - Dataset after TFIDF vectorization

1.2.3 Data Visualization

We generated some plots that might give us an insight about the data presented.

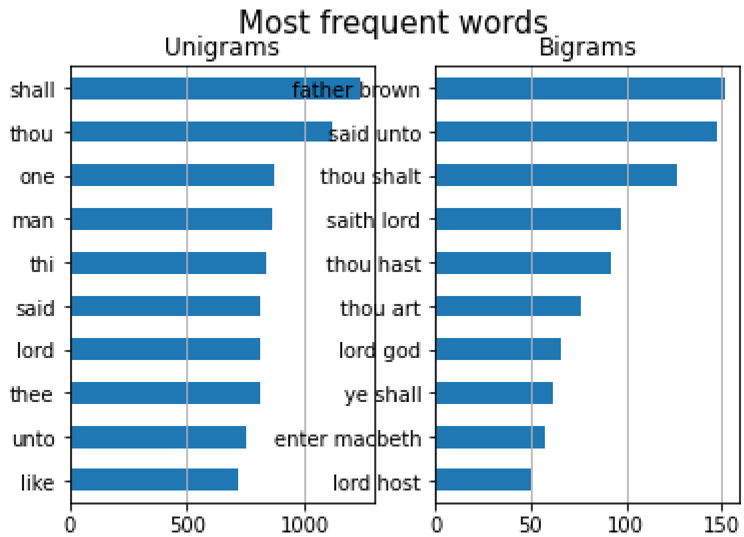


Figure 8 - Top Frequent words

Figure 9 - Wordcloud

1.3 Clustering

1.3.1 K-means

After the feature extraction step, we can now start implementing our models, but first we shuffled the data to get the best results and to make sure the model does not train on a class more than the others do.

```
# first we join the author column
author = dataFrameT["Author"]
dataset_countV = dataset_countV.join(author, rsuffix = "_")

# then we shuffle the data
dataset_countV = dataset_countV.sample(frac = 1)

# then we split the input and the target
dataset_countV_target = dataset_countV['Author']
dataset_countV_input = dataset_countV.drop(columns = ['Author'])
```

Figure 10 - Shuffling data code

Then we implemented the K-means model, we set k = 5 as we have 5 classes for 5 different authors.

First we tested with the TFIDF dataset, the first test was using the TFIDF dataset directly, we plotted the before and after clustered points.

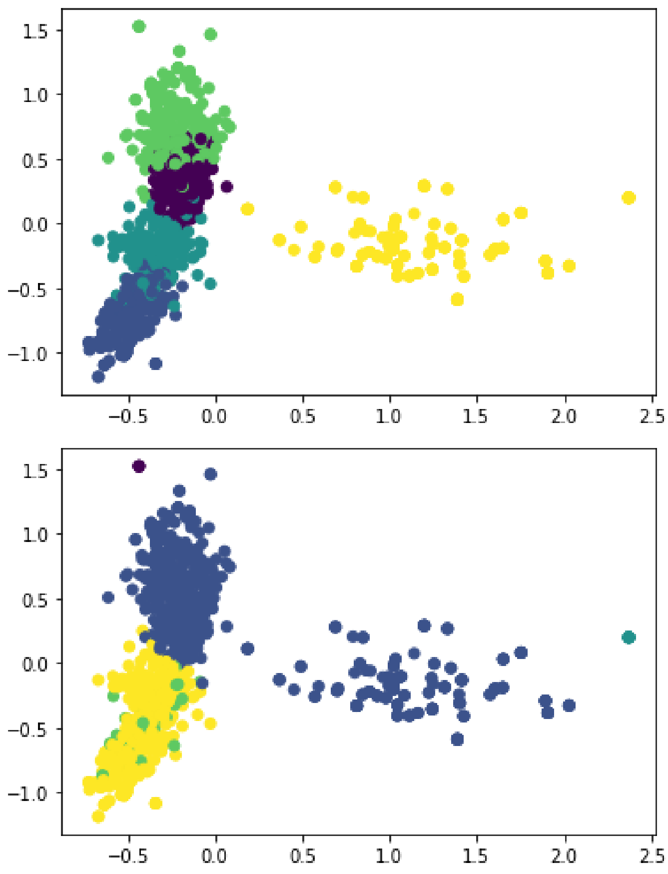


Figure 11 – Data points distribution before and after clustering

We can see that the clustering is not working very well and is not similar at all to original clustering of the data, and hence it gave a very bad score.

```
cohen_kappa score:  0.28
homogeneity score:  0.7709705820081764
Silhouette score:  -0.0434928583877836
```

Figure 12 - metric scores of k-means with TFIDF dataset

Then we plotted the data after performing PCA to reduce the dimensionality to 2, so we can plot the data and observe their distribution.

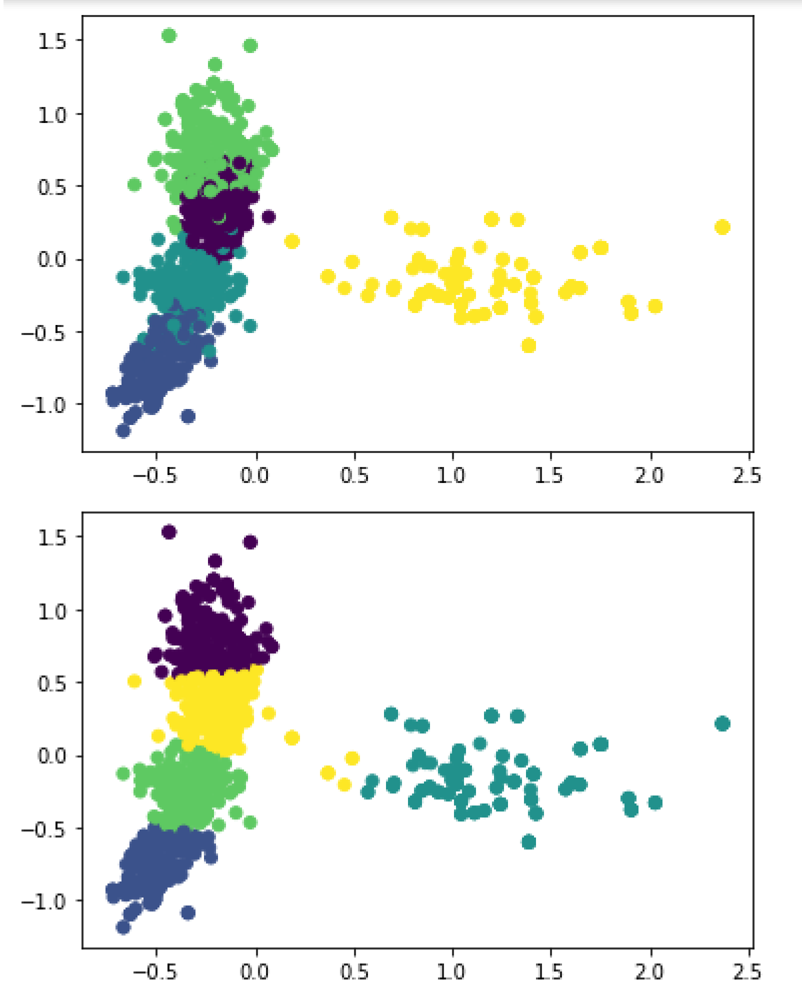


Figure 13 - Data points distribution before and after clustering with k-means model

As we can see here the clusters are very similar to the original clusters.

This gave an accuracy of

```
cohen_kappa score:  0.86875
homogeneity score:  0.7840383994132789
Silhouette score:  -0.0434928583877836
```

Figure 14 - metric scores of k-means model with TFIDF dataset and PCA

We also tested using TNSE instead of the PCA and obtained the following results.

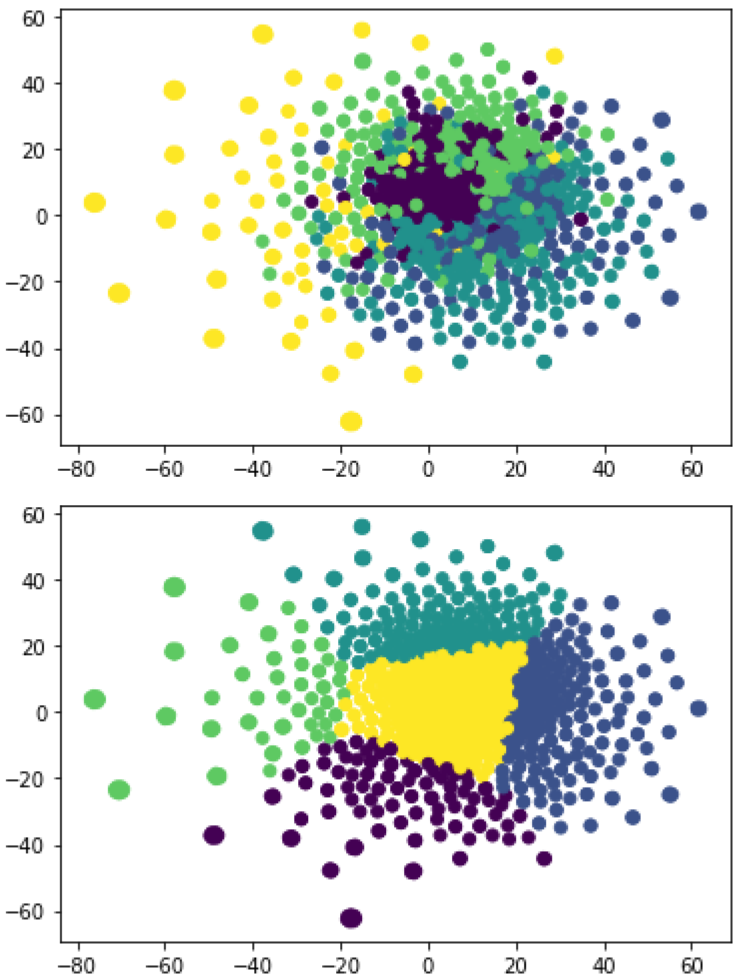


Figure 15 - Data points distribution using k-means with TSNE

Since the points aren't separable in the graph, it was expected that the accuracy acquired would be very low.

```
cohen_kappa score:  0.38249999999999995
homogeneity score:  0.2823708031991569
Silhouette score:  -0.0434928583877836
```

Figure 16 - metric scores using the k-means model with TFIDF data and TSNE

Then we tested with the BoW dataset, first without using PCA.

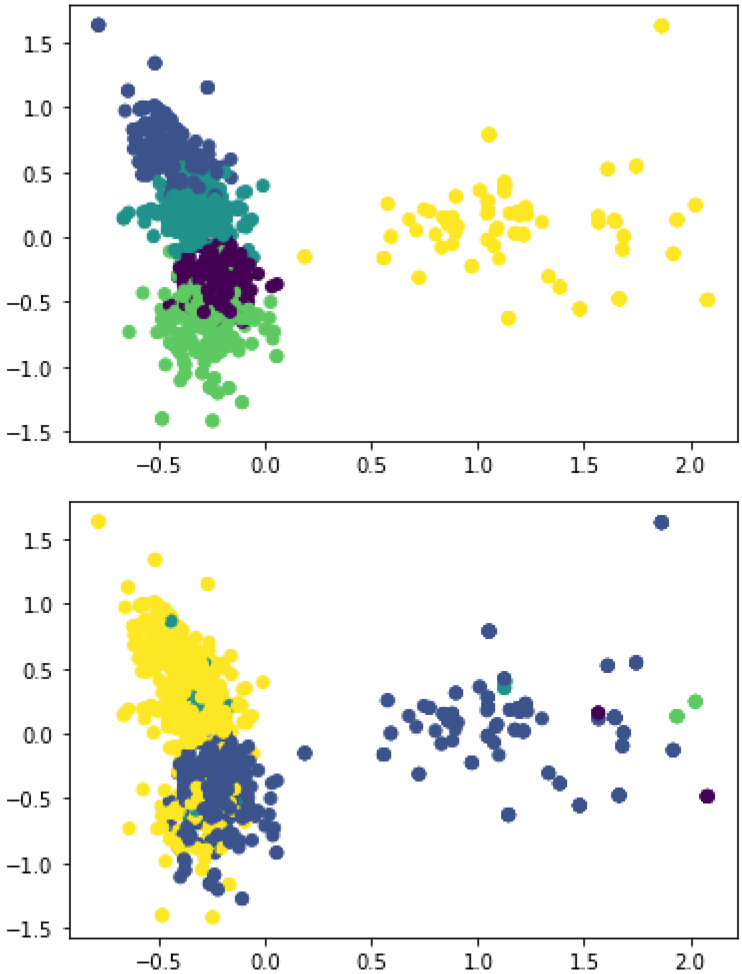


Figure 17 - Data points distribution using k-means with BoW dataset

```
cohen_kappa score:   0.14500000000000002
homogeneity score:   0.6220724070148372
Silhouette score:   -0.03335041901222637
```

Figure 18 - metric scores of k-means model with BoW dataset

Then using PCA before training the model.

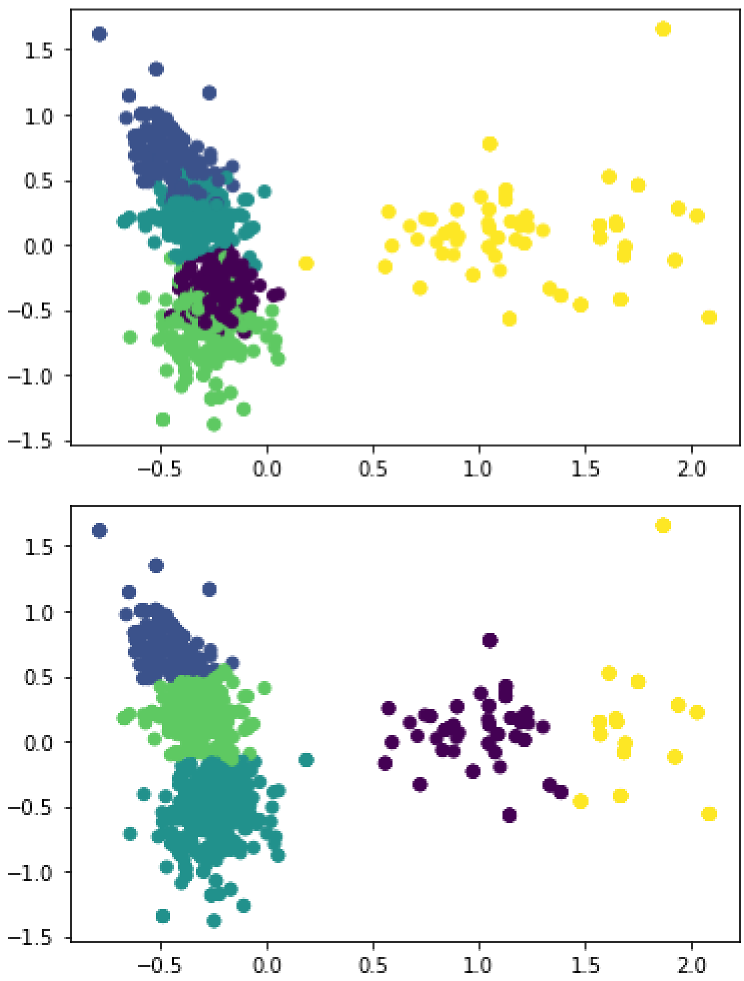


Figure 19 - Data distribution using k-means model with BoW dataset and PCA

```
cohen_kappa score:  0.36624999999999996
homogeneity score:  0.7601607339002165
Silhouette score:  -0.03335041901222637
```

Figure 20 - metric scores of k-means model with BoW dataset and PCA

Then using TSNE on the dataset before training the model.

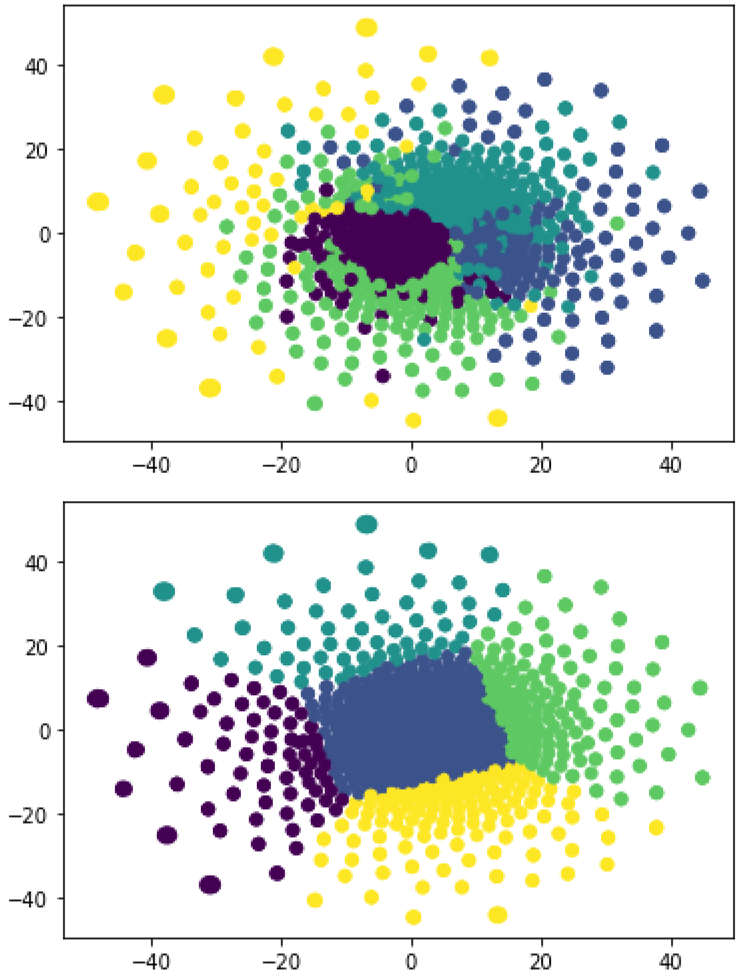


Figure 21 - Data points distribution using k-means with BoW dataset and TSNE

```
cohen_kappa score:   -0.05624999999999991
homogeneity score:  0.33100585042480407
Silhouette score:  -0.03335041901222637
```

Figure 22 - metric scores of k-means with BoW dataset and TSNE

The accuracy is very low, as it's obvious from the plot that most points got misclassified.

1.3.2 Expectation maximization

This is the plot of the original data.

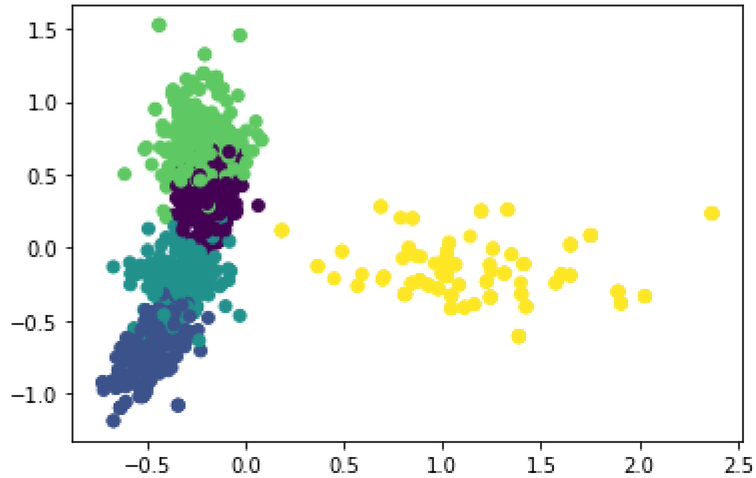


Figure 23 - original data distribution

After training the model on the TFIDF dataset, we obtained these clusters.

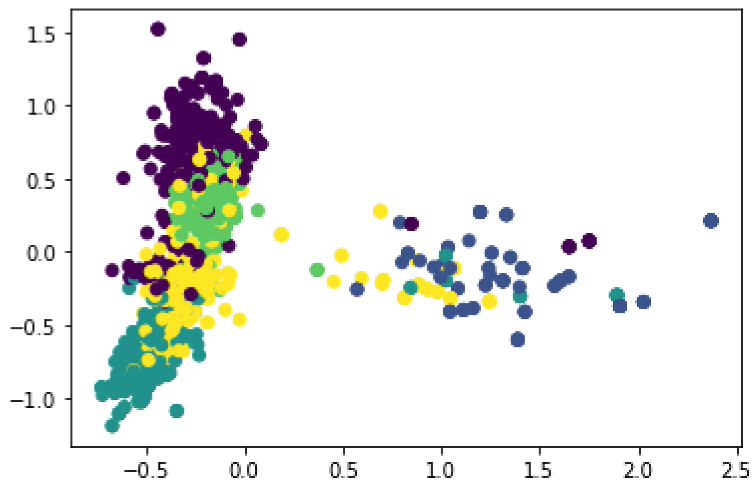


Figure 24 - data distribution after using EM with TFIDF dataset

And this gave the following accuracy

```
The kappa value is 0.6675
The Silhouttee value is -0.061833617620435544
```

Figure 25 - EM with TFIDF scores

To gain more insight on the misclassification that happened we used the cross tabulation function to plot the different classes in each cluster.

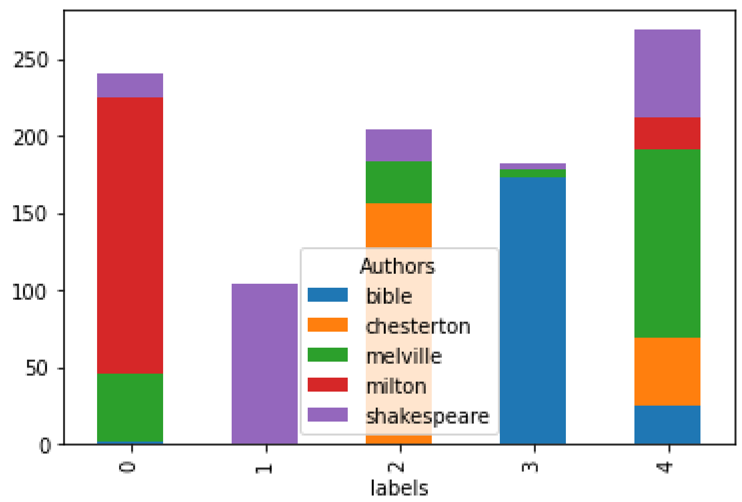


Figure 26 - cross tabulation of EM with TFIDF

So here we can see that the data in cluster zero is not purely from a single class, while in class 1 it's all from the same class. The plot explains why the kappa accuracy was high for this model.

Then we tested with the BoW dataset and obtained these results.

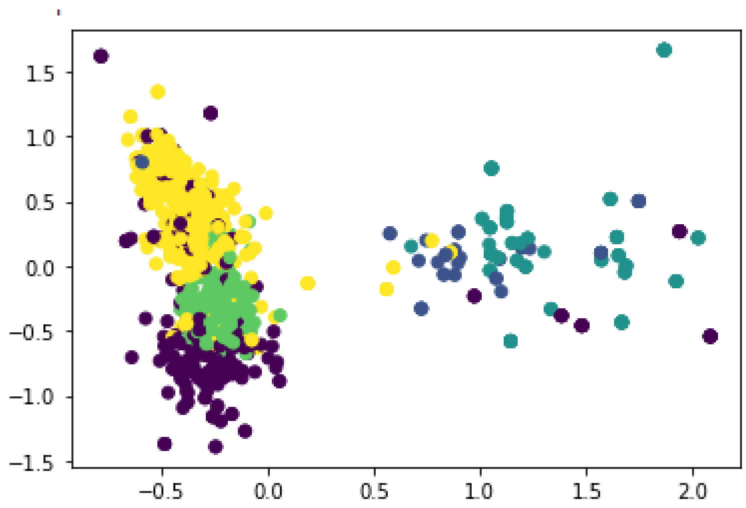


Figure 27 - data distribution using EM with BoW

```
The kappa value is 0.53625
The Silhouttee value is -0.03759544938794659
```

Figure 28 - EM with BoW dataset scores

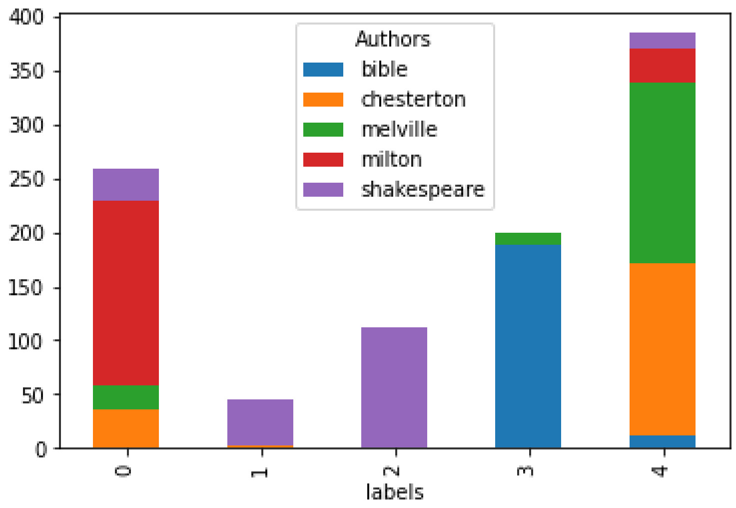


Figure 29 - Cross tabulation plot of EM with Bow Dataset

1.3.3 Hierarchical Clustering

First, we plotted the dendrogram, which shows that the best cluster number is 5 clusters.

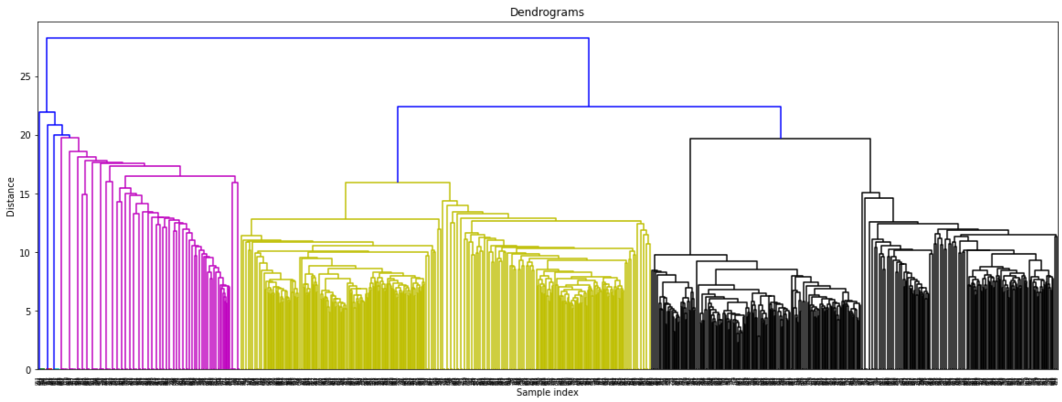


Figure 30 - Dendrogram for heirachial clustering

Then we trained the model on the TFIDF dataset.

```
homogeneity_score 0.6044286772218541
silhouette_score 0.026875126436395513
cohen_kappa_score 0.003750000000000031
```

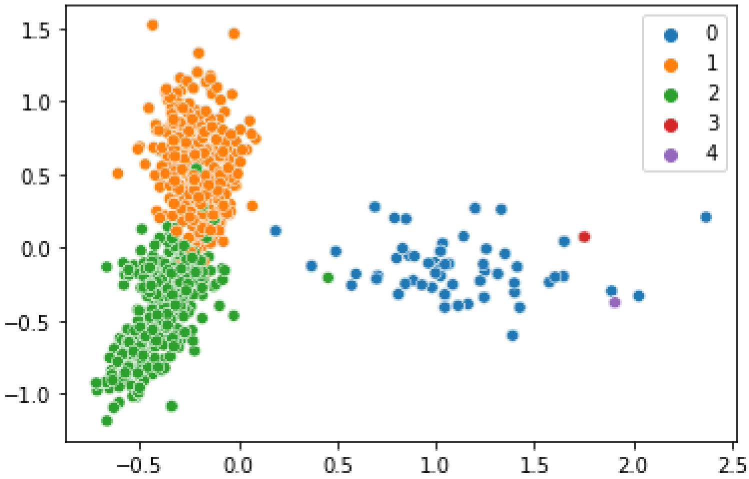


Figure 31 - Data plot and score for HC with TFIDF dataset

And then we tested on the BoW dataset.

```
homogeneity_score 0.6056077322761259
silhouette_score 0.023498537446512445
cohen_kappa_score 0.21374999999999988
```

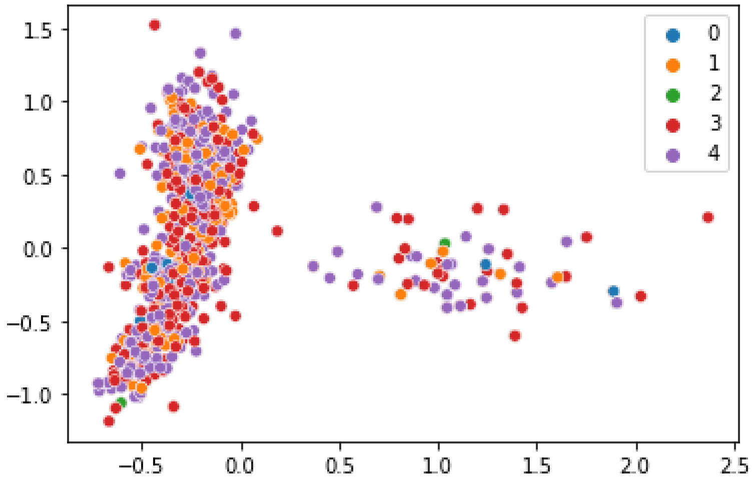


Figure 32 - Data plot and score for HC with BoW dataset

1.4 Champion Model

1.4.1 Choosing the champion model

In order to choose the champion model, we saved the silhouette and Kappa score from all the models we have tested and plotted the silhouette and kappa score on the same graph to be able to determine the point which maximizes both values.

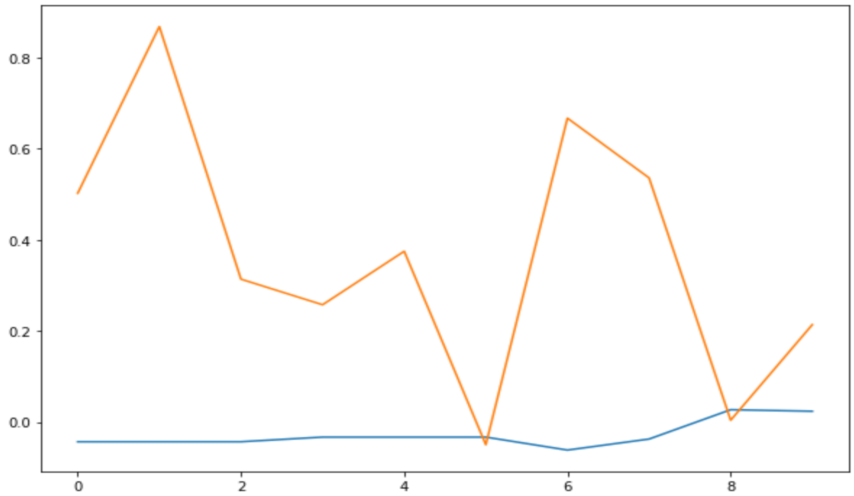


Figure 33 – silhouette and Kappa scores vs models plot

We can see that the model that has the best kappa value is the 2nd model which is the k-means with TFIDF after applying PCA, and since the silhouette values are very low and nearly non varying we will take the highest kappa score as the champion model.

We also plotted the cross tabulation plot of the winning model.

<Figure size 1080x720 with 0 Axes>

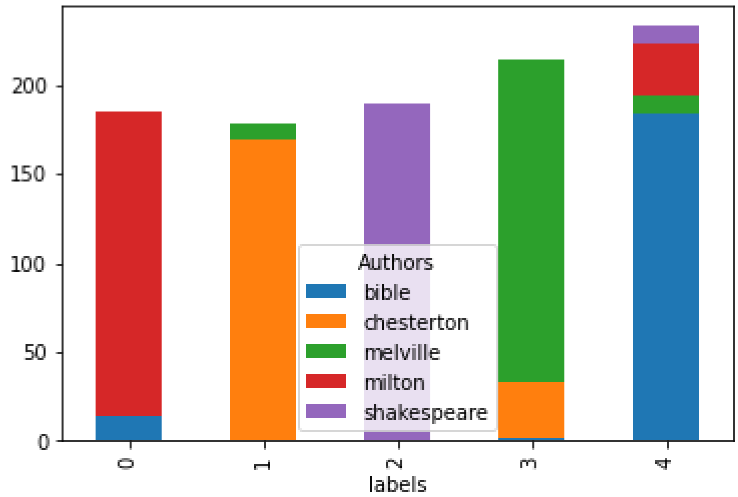


Figure 34 - cross tabulation of the champion model

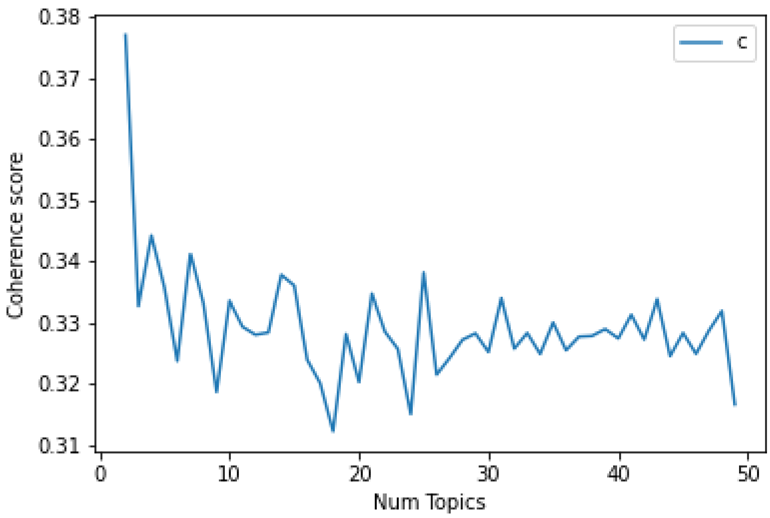
But even though this the champion model, it only achieves a kappa score of 0.87 and a very low score on the silhouette evaluation.

So we started investigating further by obtaining the wrongly classified data.

1.4.2 Error Analysis

1.4.2.1 LDA

We used the LDA (Latent Dirichlet Allocation), which is a topic modeling technique, to gain more insight about our data.
The LDA is trained over the whole documents and it starts extracting different topics from these data. We can specify the number of topics we want to extract but there is an optimal number of topics that gives the best results.
To obtain the optimal number of topics we used the coherence score as a measure to how these topics are good, so we tested the number of topics on a range from 2 to 50, plotting the coherence score of each.




```
Highest Coherence is at num topics:  2
```

Figure 35 - coherence scores for LDA model

We can see that the highest coherence score is obtained when we use only 2 number of topics. Since the coherence scores measures the degree of semantic similarity between high scoring words in the topic, then this indicates that the words are confined in only 2 topics.

To investigate these results further, we plotted the highest occurring words for each topic and we found some interesting results.

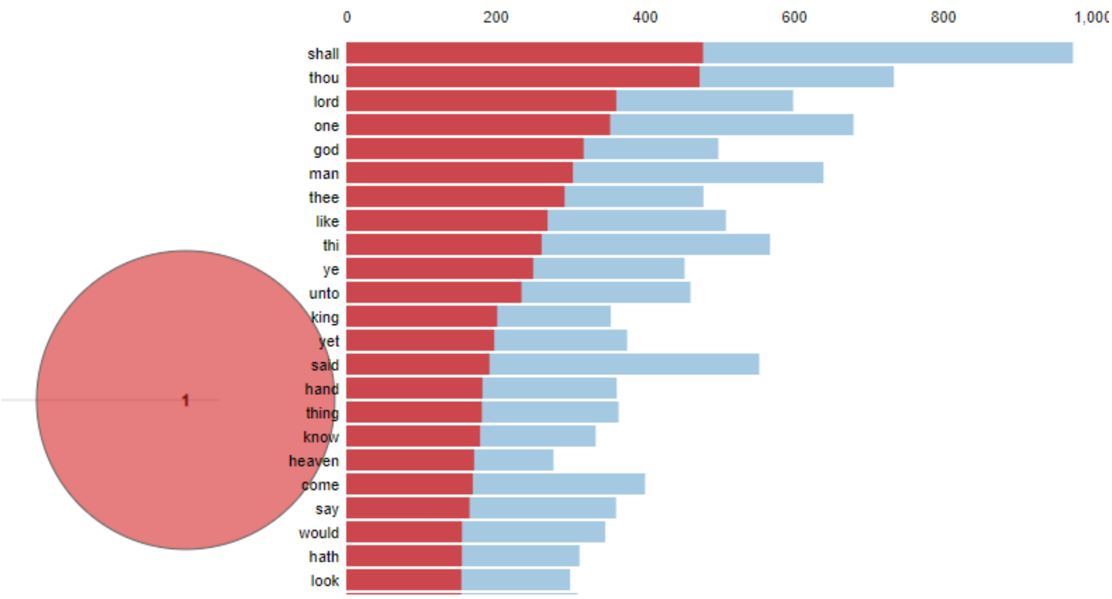


Figure 36 - Top words of importance for topic one

This is a list of the top important words in the documents, The blue line shows the total number of the word in the whole document and the red part indicates how much the first topic takes from the number of occurrences of the word.

We can notice that for topic one, the percentage it takes from the frequency of the word is nearly half.

Now plotting the second topic.

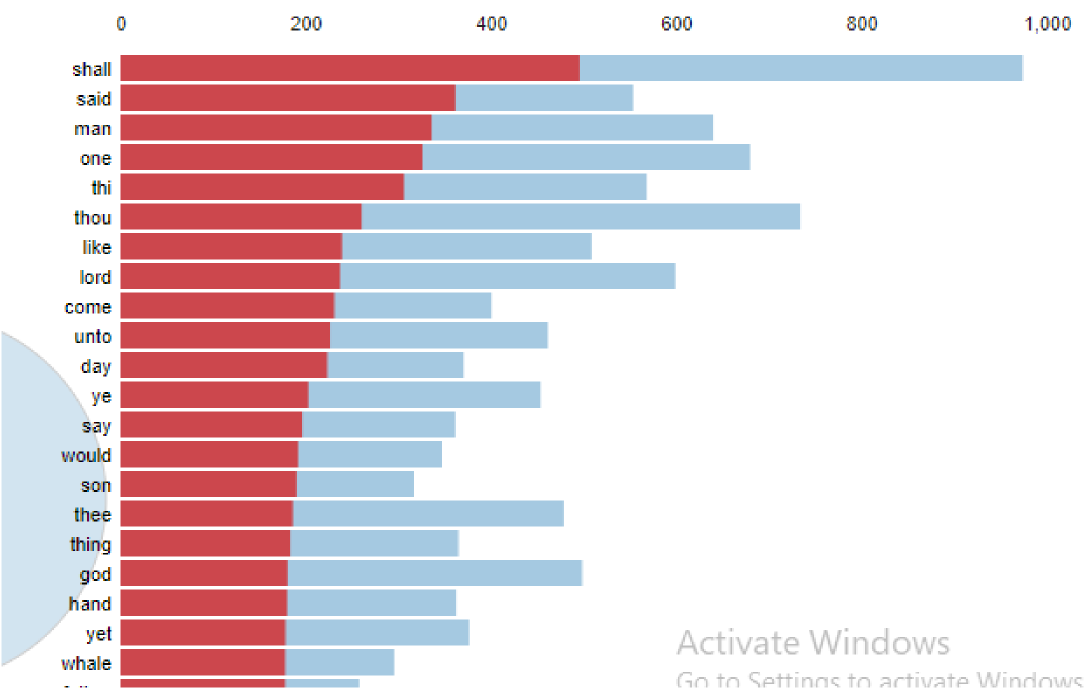


Figure 37 - Top words of importance for topic two

Here we can see nearly the same exact words as the first topic, and also the second topic takes about half of the frequency of these words.

So this can explain why the machine is unable to cluster the data in an efficient way, and this part explains why we obtained a very low silhouette score on all of the models. Since the silhouette score measures how similar an object is to its own cluster compared to other clusters, the low value indicates that the data points has more similarity with the other clusters than it should, or has equal similarity with most clusters, meaning it can't confirm that it belongs to the correct cluster.

To gain more insight on this we obtained the top 10 words for each cluster in the wrongly clustered data points.

```
Most frequent words in cluster 0
    Word  Freq
0  shall   223
1    one   190
2   thou   186
3    man   165
4   lord   157
5   said   151
6    thi   149
7   like   140
8   thee   132
9   unto   130
```

```
Most frequent words in cluster 1
    Word  Freq
0  shall   242
1   thou   184
2    one   159
3    man   158
4    thi   151
5    god   147
6   lord   142
7     ye   140
8   like   138
9   said   111
```

```
Most frequent words in cluster 2
    Word  Freq
0  shall   300
1   lord   170
2    one   165
3    man   155
4    thi   149
5   thou   149
6    god   129
7   said   125
8   like   124
9   come   123
```

```
Most frequent words in cluster 3
    Word  Freq
0  shall   165
1    one   141
2   thou   123
3    man   115
4   said   105
5   like   105
6    thi    95
7    god    90
8   king    89
9   lord    84
```

```
Most frequent words in cluster 4
    Word  Freq
0  shall   164
1    one   124
2    man   119
3   thou   117
4   unto   114
5    son   108
6   said   104
7    thi    99
8   like    82
9   lord    81
```

We can see that the words "shall", "one", "thou", "lord", "said", etc are repeated over all clusters. And we can see that these words are the same we obtained from the top frequent words over all documents.

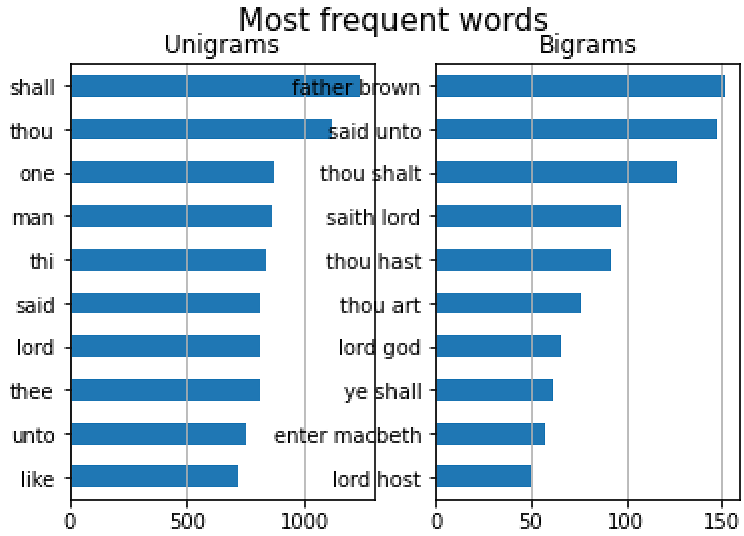
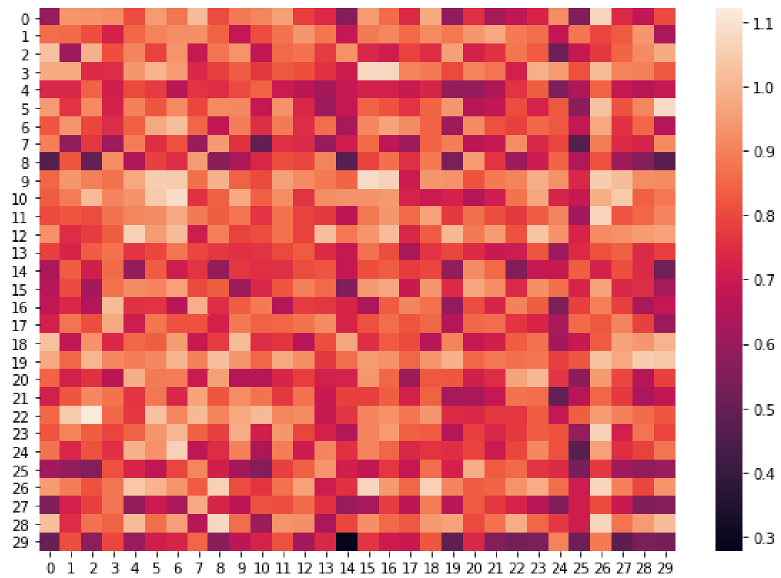


Figure 38 - Top frequent words in all documents

And the words obtained from the topic modeling using LDA plot as well, and this takes us to the main idea of choosing the books, where they are very different in terms of genre, content and semantically, but given that they revolve around the bible in different ways like writing styles or analogies and quoting from it, this causes the data to become very similar.

Another test we made using doc2vec, we compared some random documents from the bible and moby dick, and between the bible and brown books.

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fae0a3b33d0>
```



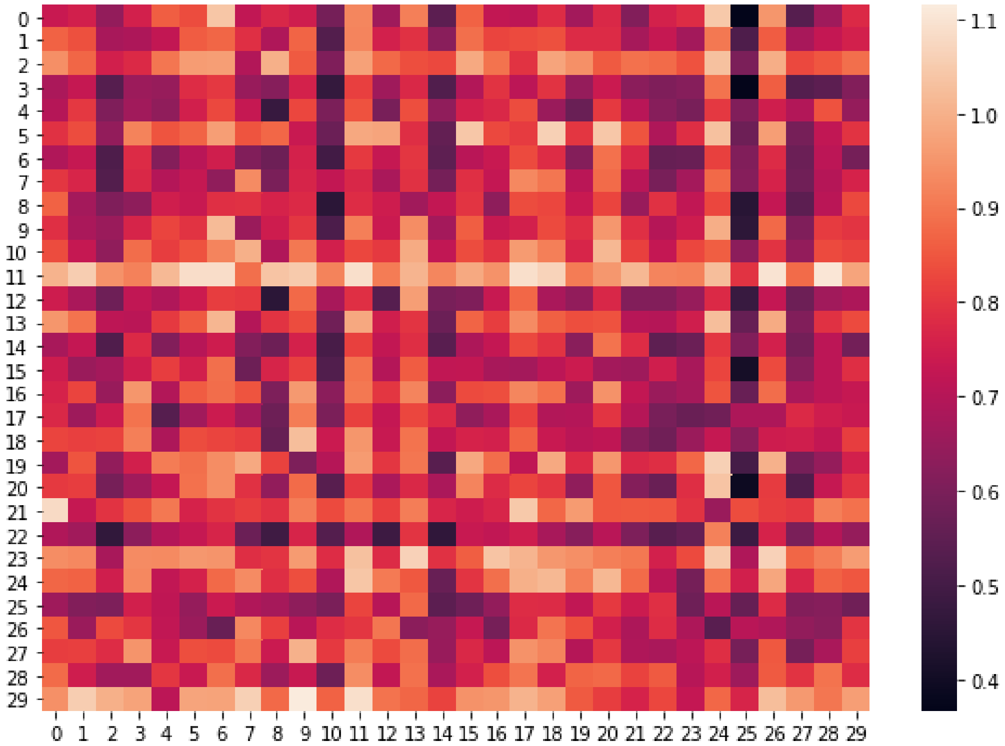
```
:] print("number of values above 0.7 in similarity score: ", counter)

number of values above 0.7 in similarity score:  725
```

Figure 39 - camparing documents bible and moby dick

And we printed the number of documents having a similarity more than 0.7, which were 725 out of 900 books

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fae09fd5950>
```




```
print("number of values above 0.7 in similarity score: ", counter2)
```

```
number of values above 0.7 in similarity score:  636
```

And as for the bible and brown books, the number of similar documents were 636 documents.

REFERENCES

https://stats.stackexchange.com/questions/375062/how-does-topic-coherence-score-in-lda-intuitively-makes-sense

https://machinelearningmastery.com/develop-word-embeddings-python-gensim/

https://cfss.uchicago.edu/notes/topic-modeling/

https://dylancastillo.co/nlp-snippets-cluster-documents-using-word2vec/

https://thinkinfi.com/gensim-doc2vec-python-implementation/