University of Ottawa
Department of Electrical and Computer Engineering

GNG 5125/ Data Science Applications

Professor: Arya Rahgozar

Project Report on Text Clustering

Submitted by

Group: DSA_202101_ 7

Yusri Al-Sanaani	300216450
Hetvi Soni	300200976
Tavleen Kour	300213090
Immanuella Iyawe	300150838

Due Date: Monday, 1st March 2021

Introduction

Clustering is a process of grouping similar items together. Each group or cluster contains items that are similar to each other. Clustering algorithms are unsupervised learning algorithms (no need for labelled datasets). In this report, three unsupervised machine learning algorithms are used:

1. K-means clustering
2. Hierarchical clustering
3. Expectation Maximization clustering

These algorithms cluster the text data taken from five different books with different genres for semantic analysis and different authors of Gutenberg digital books.

The overall objective is to produce clustering predictions and compare them; analyze the pros and cons of algorithms, use coherence coefficient, kappa analysis, silhouette score to evaluate the models and choose the best one.

The flowchart below describes the entire process we followed for clustering the text provided from the Gutenberg digital books.

We first take all five books from the digital library and do text data preparation by pre-processing the data using regex, text cleaning techniques and other functions to get the desired chunks of data. Next we perform feature engineering on the pre-processed data and provide lemmatization, stopword removal, stemming and do the label encoding as well. Now we need to convert the cleaned data into vectors using techniques such as Bag of Words, TF-iDF, Word embedding, and LDA. Then we need to reduce dimensions of the cleaned data to 2-dimensional use one of the techniques such as PCA. Next, we pass this reduced dimensioned vectors to the machine learning clustering algorithms, which are K-means clustering algorithm, Hierarchical clustering algorithm and Expectation Maximization clustering algorithm. The results we get from these algorithms can be evaluated using multiple matrices. These results can help us to decide the champion model of our clustering analysis.

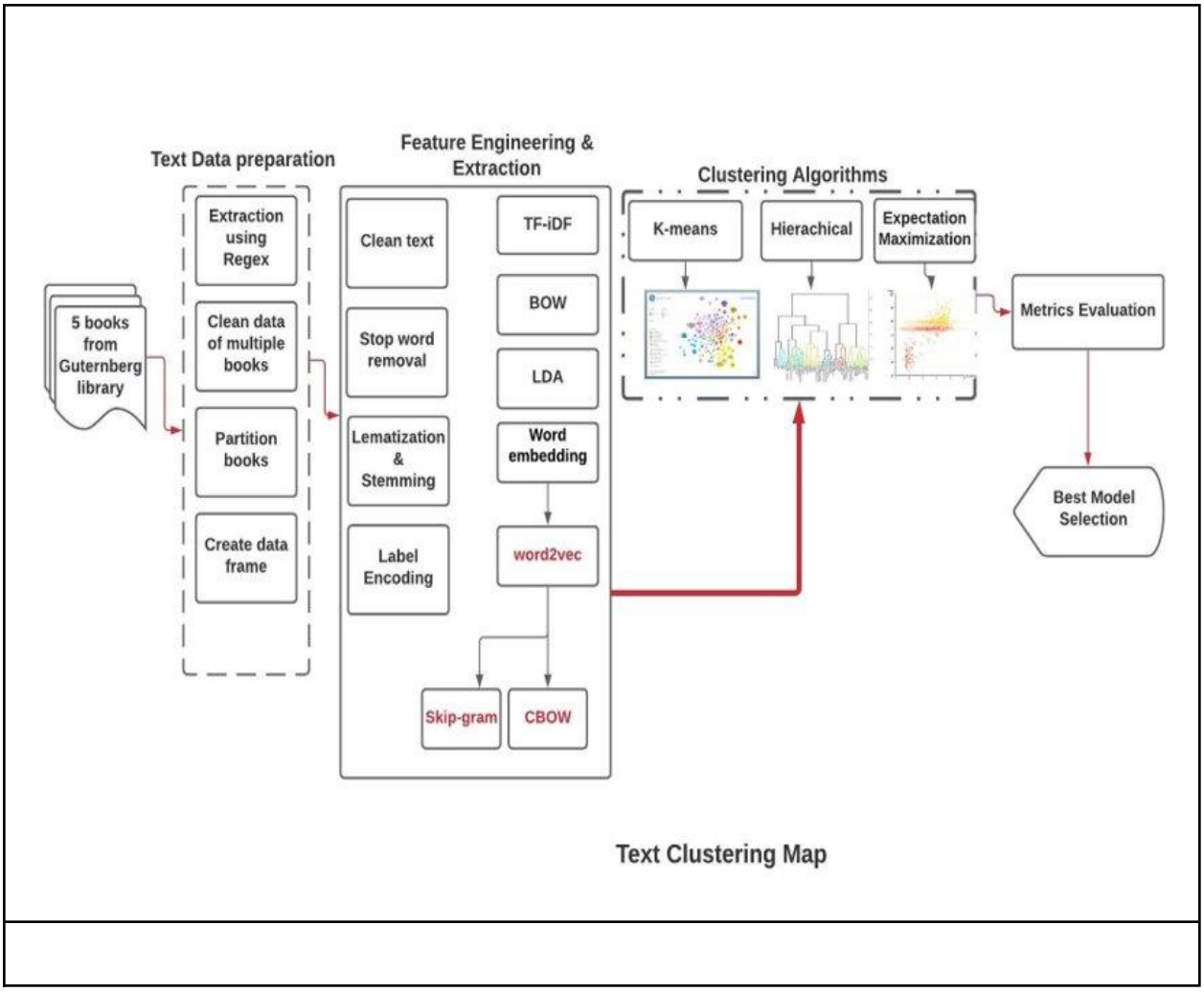

Text Data preparation
Feature Engineering & Extraction
Clustering Algorithms
5 books from Guternberg library
Extraction using Regex
Clean data of multiple books
Partition books
Create data frame
Clean text
Stop word removal
Lematization & Stemming
Label Encoding
TF-iDF
BOW
LDA
Word embedding
word2vec
Skip-gram
CBOW
K-means
Hierachical
Expectation Maximization
Metrics Evaluation
Best Model Selection
Text Clustering Map

Fig 1.1.1

1. Text Data Preparation and Pre-processing

The data set used for this project was obtained from Gutenberg's Digital library. Five different books with different authors have been imported and then pre-processed. Data preparation, data pre-processing, and text analysis will be discussed in this section.

1.1. Data preparation

For imported books, each book has been partitioned into chunks based on 150 words each. Then a set of 200 chunks has been randomly chosen from each book to create an unbiased data set. The data set was stored in a data frame. We can summarise these steps as follows:

- ➔ Import five books.
- ➔ Extract the book titles and author names using regex to get the data frame.
- ➔ Partition each book into chunks (150 words for each chunk); a function has been defined to partition multiple books. This function takes a dictionary as input that contains the labels (books and authors' names) and book content. It returns a list of partitioned books with labelling each partition to the book it belongs to.
- ➔ The code below displays the book title, author names and genres

```
# Display the books and the curresponding authors
books_authors=pd.DataFrame({'Book Title':books_titles, 'Authors':authors,'Genres':genres},index=[1,2,3,4,5])
books_authors
books_authors.to_excel("books_authors.xlsx", index=False)
```

fig1.1.2

- ➔ The genres of the books can be categorized into 5 categories:
- Metaphysical Thriller
- Children's Literature
- Poetry
- Epic poetry, Christian Mythology
- Romance Novel

- ➔ The output is displayed as shown in the data frame below.

	Book Title	Authors	Genres
1	The Man Who Was Thursday	Chesterton	Metaphysical Thriller
2	The Parent's Assistant	Edgeworth	Children's Literature
3	Leaves of Grass	Whitman	Poetry
4	Paradise Lost	Milton	Epic poetry,Christian Mythology
5	Sense and Sensibility	Austen	Romance Novel

table1.1.3

Code for creating paragraphs for 150 words each:

We divide data into chunks of 150 words each by using for loop.

```
# partitioning the text data with auto labeling
def get_chunks(chunk_books_dict):
    import nltk
    nltk.download('punkt')
    chunk_books=[]
    chunk_books_labels=[]
    for label,book in chunk_books_dict.items():
        #splits = book.split()
        tokenized_word=nltk.word_tokenize(book)
        labeled_chunks=[]
        chunks_labels=[]
        for i in range(0,len(tokenized_word),150):
            #labeled_chunks.append((' '.join(tokenized_word[i:150+i]),label))
            labeled_chunks.append(' '.join(tokenized_word[i:150+i]))
            chunks_labels.append(label)
        chunk_books.append(labeled_chunks)
        chunk_books_labels.append(chunks_labels)
    return chunk_books,chunk_books_labels
```

Fig1.1.4

Code for picking up random 200 paragraphs from each book:

Picking up 200 random chunks from each of our books can be done by calling a method known as random.sample.

```
def get_random_chunks(chunk_books):
    import random
    random.seed(1)
    random_chunks=[]
    for book in chunk_books:
        chunks=random.sample(book, 200)
        random_chunks.append(chunks)
    return random_chunks
```

Fig 1.1.5

1.2. Data Pre-Processing

After preparing the data set and storing it in a data frame. Further steps have been performed to clean the data set, perform lemmatization and stemming, and remove stop words. The figure shown below summarizes the pre-processing procedure to prepare the text data for feature engineering

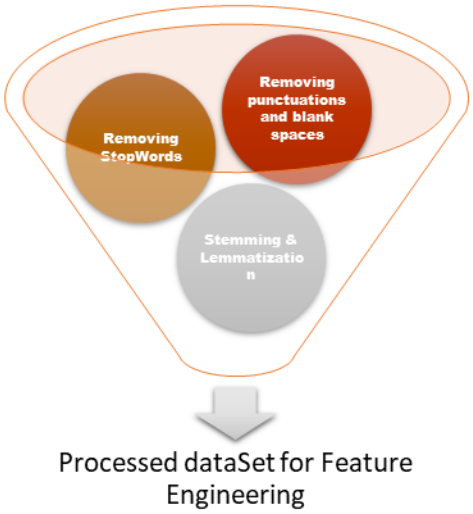


fig1.2.1

1.2.1. Data Cleaning

The data cleaning was done by creating a general function to clean the data of multiple books. This function takes a list of books as input and returns a list of cleaned books. The cleaning includes:

- ➔ Removing multiple spaces and left/right white spaces.
- ➔ Converting capital characters to lowercase.
- ➔ Removing special characters.
- ➔ Removing single character words.

Then, the cleaned data set was stored in a data frame as shown below

	books_text_data	authors_labels
395	money that we can show generosity it is by giv...	Edgeworth
939	as her own and with a mind tormented by self r...	Austen
842	his marriage with miss grey it was no longer t...	Austen
367	halfpence no indeed it is not cried paul it is...	Edgeworth
567	as these and abreast with them prudence the la...	Whitman

table 1.2.2

1.2.2. Stop Words Removal

The data samples from the book are then prepared for further process using nltk library to remove the stop words. The figure below shows the data frame for the data set after removing stop words.

<u>Code for removing the stopwords:</u>

The below code displays the standard code to remove all the stopwords from a language. We choose English since all of our books are in that language.

```
# remove Stopwords
def remove_stopwords(books_text_data):
    nltk.download('stopwords')
    stop_words = nltk.corpus.stopwords.words("english")
    splited_text=books_text_data.str.split()
    text_no_stopwords=[]
    for i in range(len(splited_text)):
        text_no_stopwords1 = [word for word in splited_text[i] if word not in stop_words]
        text_no_stopwords.append(' '.join(text_no_stopwords1))
    return text_no_stopwords
```

fig1.2.3

After removing stopwords:

The diagram below displays the dataframe which has been updated after stop words were removed from the text.

	books_text_data	authors_labels	text_no_stopwords
893	she added that when the money is once parted w...	Austen	added money parted never return sisters marry ...
216	at the door yet it was his fate once to be sen...	Edgeworth	door yet fate sent message unlucky time door h...
382	kept it at home as she should have done perhap...	Edgeworth	kept home done perhaps careful received strong...
484	myths of the greeks and the strong legends of ...	Whitman	myths greeks strong legends romans hear tale d...
817	esteemed him the more because he was slighted ...	Austen	esteemed slighted willoughby marianne prejudic...

table1.2.4

1.2.3. Lemmatization and Stemming

Next, we removed all the similar root words using the lemmatization function from the nltk library to avoid having multiple features for the same root word (e.g.: cats and cat). We further create a column containing data after lemmatization and stop words removal. We also tried to remove all the suffixes from the tokenized words using the stemming function from the nltk library.

Code to perform lemmatization and stemming:

The below code displays the standard code to perform lemmatization and stemming.

```
# Lemmatisation (convert the word into root word)
def lemmatisation(books_text_data):
    nltk.download('wordnet')
    lem = nltk.stem.wordnet.WordNetLemmatizer()
    splited_text=books_text_data.str.split()
    lemmatized_words=[]
    for i in range(len(splited_text)):
        lemmatized_words1 =' '.join([lem.lemmatize(word) for word in splited_text[i]])
        lemmatized_words.append(lemmatized_words1)
    return lemmatized_words

#Stemming
def stem_word(books_text_data):
    ps=nltk.stem.porter.PorterStemmer()
    splited_text=books_text_data.str.split()
    stem_words=[]
    for i in range(len(splited_text)):
        stem_words1 =' '.join([ps.stem(word) for word in splited_text[i]])
        stem_words.append(stem_words1)
    return stem_words
```

fig1.2.5

After performing lemmatization and stemming:

The below diagram displays the dataframe which has been updated after lemmatization and stemming has been performed to the text.

	books_text_data	authors_labels	labels_encoder	text_no_stopwords	lemmatized_text	lemm_nostopwords_text	stem_text
186	regarded as barbaric and as involving cruelty ...	Chesterton	1	regarded barbaric involving cruelty cow cruelt...	regarded a barbaric and a involving cruelty to...	regarded barbaric involving cruelty cow cruelt...	regard as barbar and as involv cruelti to the ...
599	wondrous house that delicate fair house that r...	Whitman	4	wondrous house delicate fair house ruin immort...	wondrous house that delicate fair house that r...	wondrous house delicate fair house ruin immort...	wondrou hous that delic fair hous that ruin th...
420	my destination i understand your anguish but i...	Whitman	4	destination understand anguish help approach h...	my destination i understand your anguish but i...	destination understand anguish help approach h...	my destin i understand your anguish but i can ...
570	oratorios of beethoven handel or haydn the cre...	Whitman	4	oratorios beethoven handel haydn creation bill...	oratorio of beethoven handel or haydn the crea...	oratorio beethoven handel haydn creation billo...	oratorio of beethoven handel or haydn the crea...
251	knew had usually a great effect even at thirte...	Edgeworth	2	knew usually great effect even thirteen observ...	knew had usually a great effect even at thirte...	knew usually great effect even thirteen observ...	knew had usual a great effect even at thirteen...

table 1.2.6

2. Feature Engineering

We perform the following functions under feature engineering:

1. Label Encoding
2. Transformation models such as:
    a. Bag of Words
    b. TF-iDF
    c. LDA(using gensim and sklearn libraries)
    d. Word Embedding(word2vec and doc2vec)

2.1. Label Encoding

The authors labelled have been encoded to be transformed to a numerical value between 0 and n_{label}-1.

Code for Label Encoding:

We pass our text from the dataframe to a function called LabelEncoder() from sk-learn library.

```
# Encode the authors labels
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
labels_encoder=le.fit_transform(df_books['authors_labels'])
```

fig2.1

2.2. Bag of Words (BOW) Transformation

Bag of Words is a natural language processing text modelling technique. We cannot feed our text directly into algorithms, so we convert our text into numbers. This technique is used to process the text data into the bag of words which keeps a count on the total numbers of words used in the entire text. We can visualize the data by displaying the number of word counts mapping against the words.

For BOW feature extraction we made use of the function available in the ScikitLearn Library. In BOW, the countVectorization converts the number of repeated words into a sparse matrix which can be used further for modelling the data.

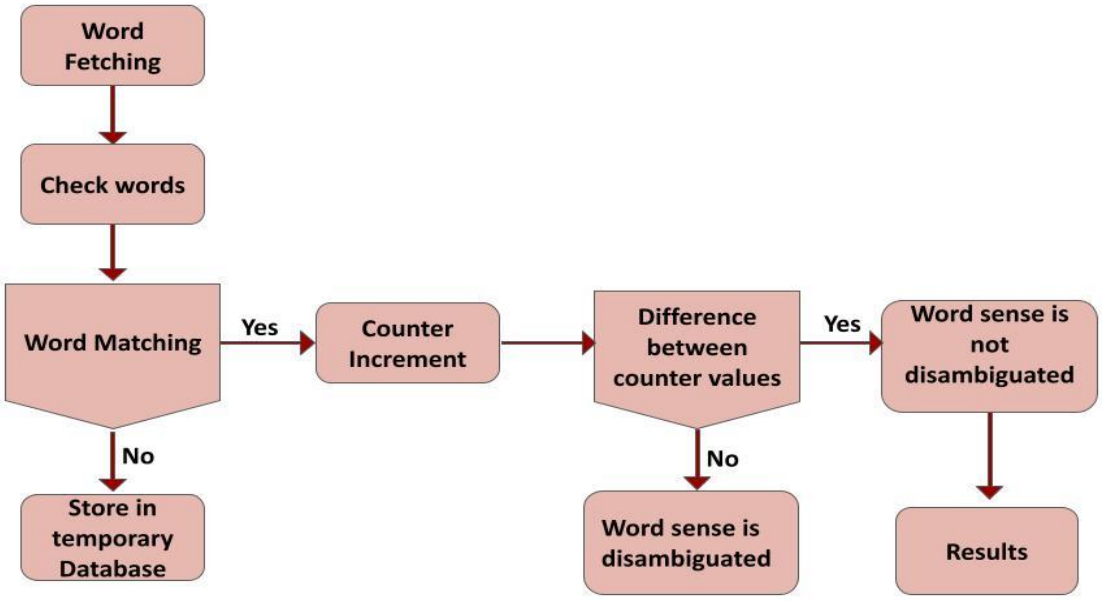
Advantages of BOW:

- ➔ The main advantage of bag of words is that it is very easy to implement and understand.
- ➔ This technique has a lot of flexibility for customization of the dataset.
- ➔ It is used on prediction problems like documentation classification and language modelling.

Disadvantages of BOW:

- ➔ If the same word is seen in upper case and lower case, BOW generates different tokens for them.
- ➔ Bag of words assumes that all words are independent of each other.
- ➔ BOW has low accuracy for sentiment analysis due to ignorance of the grammar and semantics of the words.

fig2.2.1



The above diagram shows the process of implementation of the bag of words. Here the words are fetched at first from the text and then are checked to see if they match with any existing words in the model. If not then they are stored separately with counter value of 1, and if they do, the counter increases by 1 for that word. If now the counter values are the same for multiple words then we infer that the word doesn't match the interpretation of another word and if the counter is different for words then that word is added as a result.

Code for Bag of Words:

For Bag of Words Implementation, we simply use CountVectorizer() function from the sci-kit learn library as shown below

```
#Transformation to bag of words
from sklearn.feature_extraction.text import CountVectorizer
BOW= CountVectorizer().fit_transform(x).toarray()
```

fig2.2.2

2.3. TF-iDF Transformation

TF-iDF stands for term frequency-inverse document frequency. It is a statistical technique that reflects how important a word is in a document. For TF-iDF, TfidfVectorizer() function was used from the Sci-kit learn library. TF-iDF sums up the number of word occurrences in the document with the number of times it appears in the document.

Term Frequency: TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document)

Inverse Document Frequency: IDF(t) = log_e(Total number of documents / Number of documents with term t in it)

Advantages of TF-iDF:

- ➔ TF-iDF techniques result in a document that extracts the most descriptive words present inside a document.
- ➔ It can be used to measure the unique and relevant data in the document.
- ➔ It can be calculated easily.

Disadvantages of TF-iDF:

- ➔ Semantic similarities between words are not used at all.
- ➔ This technique is slow for large data as it computes the document similarity directly in the word- count space.
- ➔ It assumes that the counts of each word provide an independent verification of resemblance.

Code for TF-iDF:

For TF-iDF, we used the TfidfVectorizer() function from the sklearn library and also selected various parameters as per our requirements like assigning n-gram range as shown below.

```
from sklearn.feature_extraction.text import TfidfVectorizer
TFiDF = TfidfVectorizer() # I have reduce the # of features to 200 to reduce the runtime
TF_iDF = TFiDF.fit_transform(x).toarray()
```

```
#TFiDF Transformation
# Parameter election
from sklearn.feature_extraction.text import TfidfVectorizer
ngram_range = (1,2); min_df = 10; max_df = 1.
#; max_features = 300
# max_features=max_features
tfidf = TfidfVectorizer(encoding='utf-8',
                        ngram_range=ngram_range,
                        stop_words=None,
                        lowercase=False,
                        max_df=max_df,
                        min_df=min_df,
                        norm='l2',
                        sublinear_tf=True)
TFiDF= tfidf.fit_transform(x).toarray()
TFiDF.shape
```

fig2.3

2.4. Word Embedding

Word Embedding is a model /technique which can be used for mapping word/text into a vector space, such that they appear in high dimensional vector space. From that vector, one can easily understand the similarities and dissimilarity between words in a given Data set.

Word embedding can be implemented in 3 different ways

- One Word Context (CBOW)
- Multi-Word Context (Skip Gram)
- Global Vectors for Word Representation (GloVe)

For our clustering algorithms, we have tried to implement the Skip-Gram method of Word embedding using Scikit Learn Library.

Advantages:

- ➔ Probabilistic in nature.
- ➔ Requires less memory. It requires less RAM for computation.
- ➔ Skip Gram performs better even with large Data sets.

Disadvantages:

- ➔ CBOW performs smoothly when the Data Set is small.
- ➔ Scaling to new languages requires a new Embedding matrix.
- ➔ Inability to handle new or unknown words.

Code for Word2Vec:

The below code displays the implementation of a continuous bag of words and skip-gram for word embedding. We implemented these by using methods from the gensim library.

```
from gensim.models import Word2Vec
books_text = x.values
textVec = [nltk.word_tokenize(text) for text in books_text]
#continuous bag of words
cbow_model = Word2Vec(textVec, min_count=1)
#skip-gram
sg_model = Word2Vec(textVec, min_count=1, sg=1)
words1 = list(cbow_model.wv.vocab)
words2 = list(sg_model.wv.vocab)
```

fig2.4

2.5. Latent Dirichlet Allocation

Topic modelling means a statistics model which identifies the abstract topics present in a document.
Latent Dirichlet Allocation is one of the most popular topic modelling techniques. LDA assumes that the document comprises multiple words that help map the document to a list of topics by assigning each word to different topics. In the assignment, we used the LDA method with both gensim and scikit-learn libraries.

Advantages:

- ➔ Discovers hidden topic patterns.
- ➔ LDA can be implemented on complicated documents.
- ➔ Reduces overfitting better than other topic models.

Disadvantages:

- ➔ Ignores the order of the words and their syntactic information.
- ➔ The clustering is non-hierarchical.
- ➔ We sometimes get results on totally unrelated topics as it can not capture correlations.

Code for LDA using scikit-learn library:

Here we implemented LDA using the sk-learn library. We define n_components here as the number of topics we want to pass to our model.

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.decomposition import LatentDirichletAllocation
lda_model_s=LatentDirichletAllocation(n_components=5,learning_method='online',random_state=0,n_jobs=-1)
lda_output=lda_model_s.fit_transform(BOW)
print(lda_output)
print(lda_output.shape)
```

fig 2.5.1

Code for LDA using gensim library:

Now we implemented LDA using the gensim library. We need to pass multiple parameters for this model such as number of topics, chunk size, number of iterations etc.

```
import gensim
from gensim import corpora
training_set = (x).tolist()
training_set = ' '.join(map(str, training_set))
nltk_tokens = nltk.word_tokenize(training_set)
dataset = [d.split() for d in nltk_tokens]
dictionary = corpora.Dictionary(dataset)
doc_term_matrix = [dictionary.doc2bow(rev) for rev in dataset]
```

```
#Now we create LDA model using Gensim
# Creating the object for LDA model using gensim library
LDA = gensim.models.ldamodel.LdaModel
# Build LDA model
lda_model = LDA(corpus=doc_term_matrix, id2word=dictionary, num_topics=10,
                random_state=100,chunksize=2000, passes=20,iterations=400)
```

Fig 2.5.2

Visualization of LDA model using gensim library:

➔ **When the number of topics is 10:**

We notice that we topics are overlapping each other since the number of topics is very high considering our data. We can get better results by reducing the number of topics.

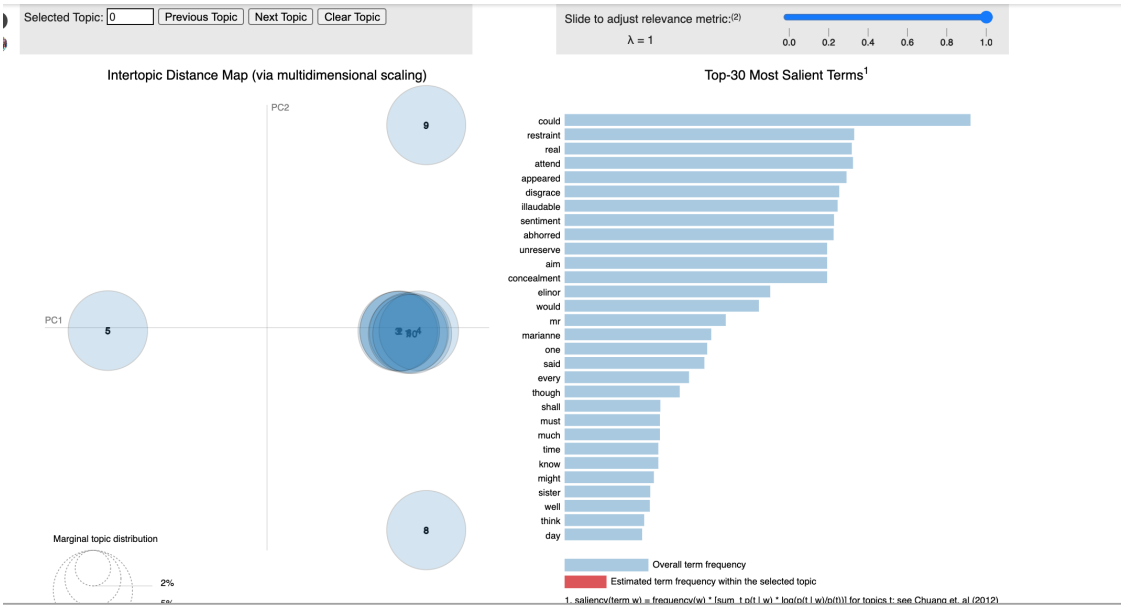


Fig 2.5.3

➔ **When the number of topics is 4:**

When now we implement the same model but the number of topics is reduced to 4, we get better results than the previous analysis. One way to check the ideal number of topics to pass to our LDA model is by calculating the coherence score which is discussed in later stages in the report.

The first picture depicts the overall topics and top words for all of them.

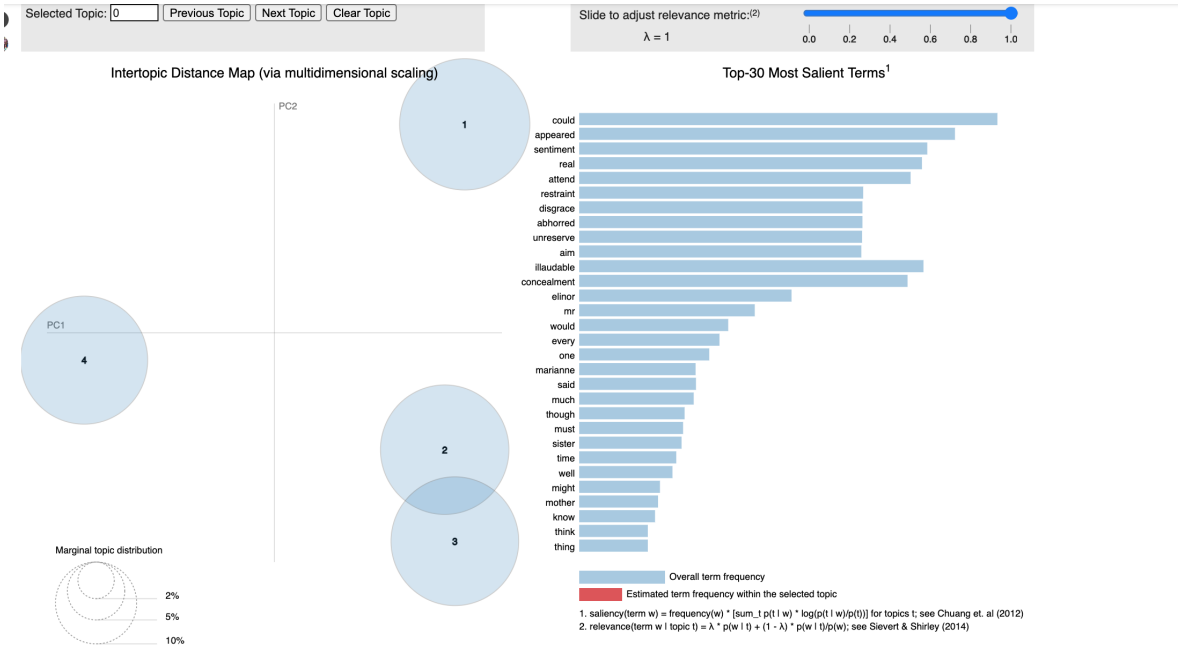


Fig 2.5.4

2.6 Tackling Multidimensional Data

Data visualization is only possible for 2 or 3-dimensional data. To visualize data that is multiple-dimensional we need to reduce its dimensions using some techniques. We need to specify the number of dimensions `n` to which we want to transform our data. This technique is required for clustering algorithms for visualization as the vectors we get after feature engineering is multidimensional(more than 3 dimensions)

Common techniques for this process are:

- ➔ Principal Component Analysis commonly known as PCA.
- ➔ t-Distributed Stochastic Neighbor Embedding commonly known as t-SNE.

1. **Principal Component Analysis**:

One of the most common and widely used techniques for reducing dimensions is PCA where we specify the number of dimensions to which we want to transform our data and pass it to the algorithm.

In the graph displayed below, we can notice that the clusters formed as a result of PCA are well scattered but are somewhat overlapping each other.

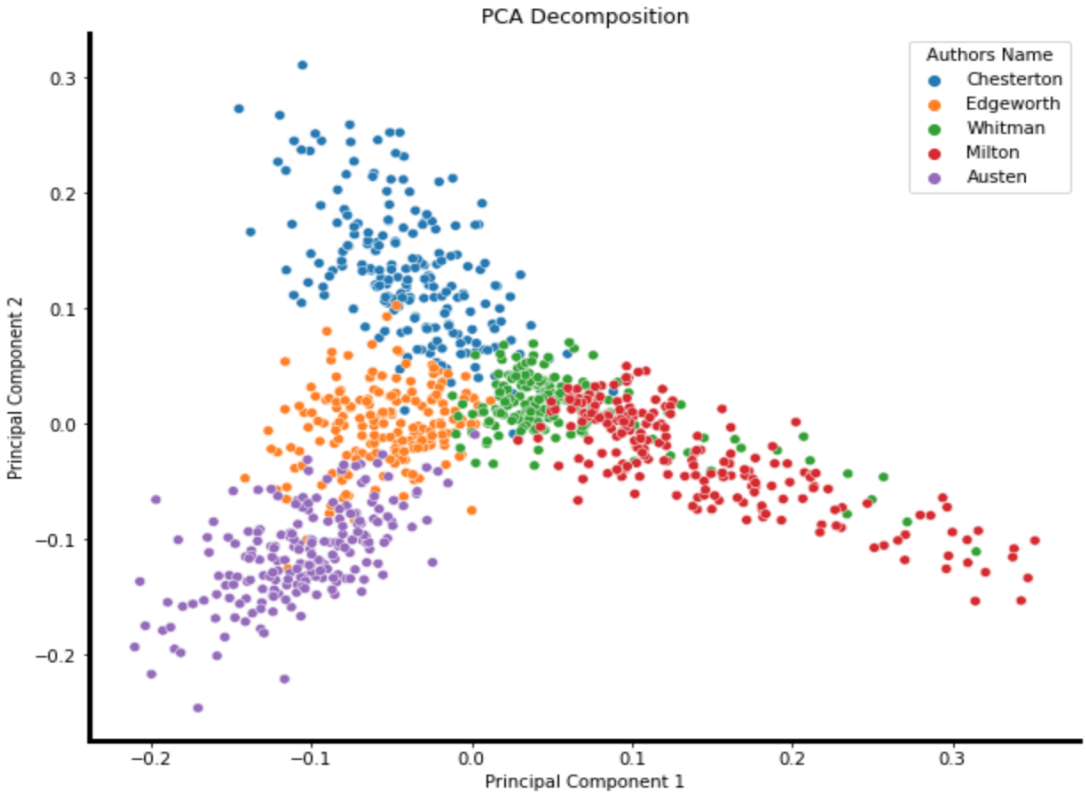


Fig2.6.1

2. **<u>t-Distributed Stochastic Neighbor Embedding:</u>**

The technique follows a procedure of giving each datapoint a location 2 or 3-dimensional graph. The plotted points are in such a way that similar words are plotted at close points and the dissimilar words are plotted more scattered.

In the graph visualized below, we can see five distinct clusters formed by t-SNE with some overlapping.

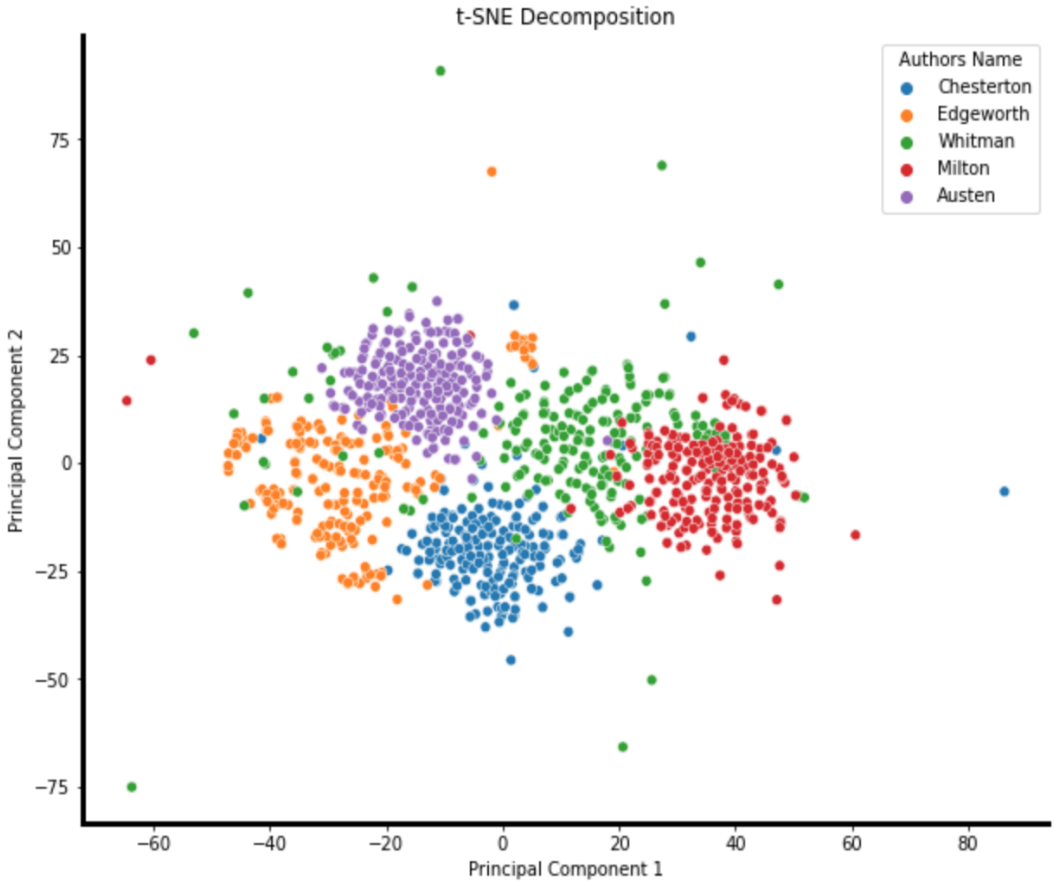


fig2.6.2

2.7 Cosine Similarity

Cosine similarity measures the similarity between two vectors by calculating the distance between them. This uses the mathematical calculations of taking angle between two vectors and calculating the cosine of that angle. This is majorly used to define how similar two documents are irrespective of their size.

The graph below visualizes the cosine similarity of our data using the bag of words vectorization technique.

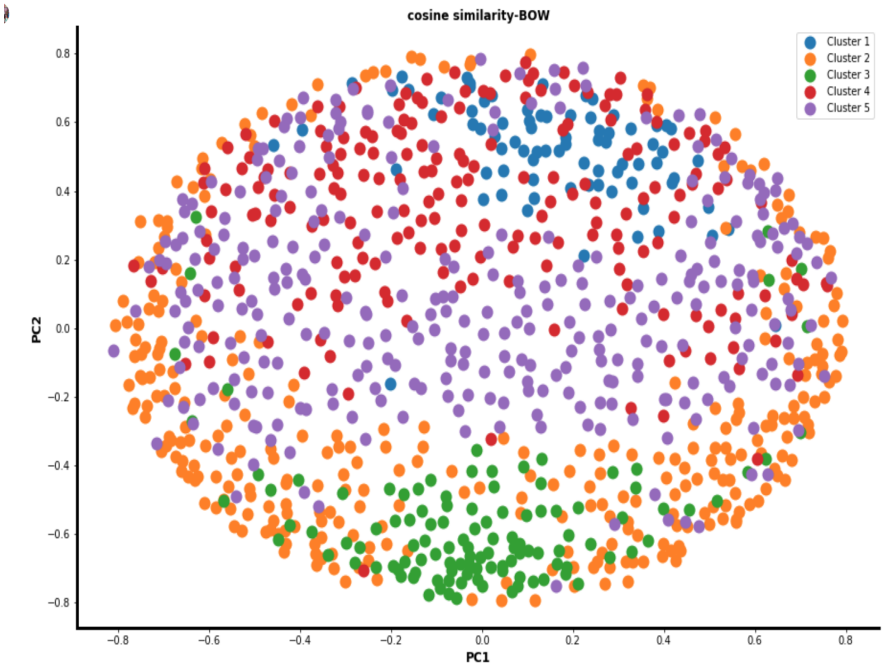

cosine similarity-BOW
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
PC1
PC2
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8

fig2.7

3. Clustering Models

Clustering is a technique to combine similar data points into different groups. Each such group is known as a cluster.
We implemented three machine learning clustering models for unsupervised learning and compared them to find the champion model. The three models are listed below:

1. K-means clustering
2. Hierarchical clustering
3. Expectation Maximization clustering

The algorithms are implemented using Scikit learn libraries. The methodology used to train each model is as follows:

- ➔ Convert text into vector space.
- ➔ This is multidimensional data. Reduce the dimensions to two using PCA(Principal component analysis)
- ➔ Pass the results from PCA to the clustering algorithms.
- ➔ Now, plot the clustering algorithm analysis on a scatter plot.
- ➔ Analyse the results based on coherence analysis, kappa coefficient, silhouette score.

3.1. K-means Clustering

Many clustering algorithms are available in the scikit learn library and elsewhere, but k-Means is the easiest to understand and implement.
It is an unsupervised clustering algorithm.
The *k*-means algorithm searches for a predetermined number of clusters within an unlabeled multidimensional dataset. It accomplishes this using a simple conception of what the optimal clustering looks like:

• The “cluster center” is the arithmetic mean of all the points belonging to the cluster.
• Each point is closer to its cluster centre than to other cluster centres.

The most crucial step for any unsupervised learning algorithm is to determine the optimal number of clusters into which the data may be clustered.
The value of k can either be manually assigned or can be chosen using an analytical method there are two methods: Elbow method and Silhouette analysis.

Elbow method:

The Elbow method is a technique that is used to determine to find the number of clusters for the given DataSet
The method runs K-means Algorithm on a given dataset for a predefined number of times (say between 1-10) and then for each k value, an average score is calculated for all the clusters, it is the Distorted score that is computed.
The value is then plotted in a line chart form, then the value of k is chosen for the number where the line plot bends a little.
This obtained value is assumed to be a good choice for the k value and it produces good clustering results.

Elbow Method Implementation:

Step1: For Elbow Method implementation we have modelled the k means algorithm to be implemented 10 times the error score is been calculated as shown below:

```
from sklearn.cluster import KMeans
wcss=[]
for i in range(1,10):
    kmeans=KMeans(n_clusters=i)
    kmeans.fit(PCA_components.iloc[:,:4])
    wcss.append(kmeans.inertia_)
```

```
plt.figure(figsize=(10, 7))
plt.plot(range(1,10),wcss,'b-',lw=3.0)
plt.title('The Elbow Method',fontsize = 12,fontweight='bold')
plt.xlabel('Number of Clusters',fontsize = 12,fontweight='bold')
plt.ylabel('WCSS',fontsize = 12,fontweight='bold')
#plt.show()
import matplotlib as mpl
mpl.rcParams["axes.spines.right"] = False; mpl.rcParams["axes.spines.top"] = False
from pylab import *
rc('axes', linewidth=3)
fontsize = 12
ax = gca()
for tick in ax.xaxis.get_major_ticks():
    tick.label1.set_fontsize(fontsize)
    tick.label1.set_fontweight('bold')
for tick in ax.yaxis.get_major_ticks():
    tick.label1.set_fontsize(fontsize)
    tick.label1.set_fontweight('bold')
plt.show()
```

fig3.1.1

Step2: Visualization of line plot for Elbow Method.
From the graph below we can see that the plot bends at the value = 3, so we can choose k = 3 for optimal cluster results for k-means clustering algorithm. But if we notice closely we can see that we also see slight bends at k = 4 and k = 5 as well.

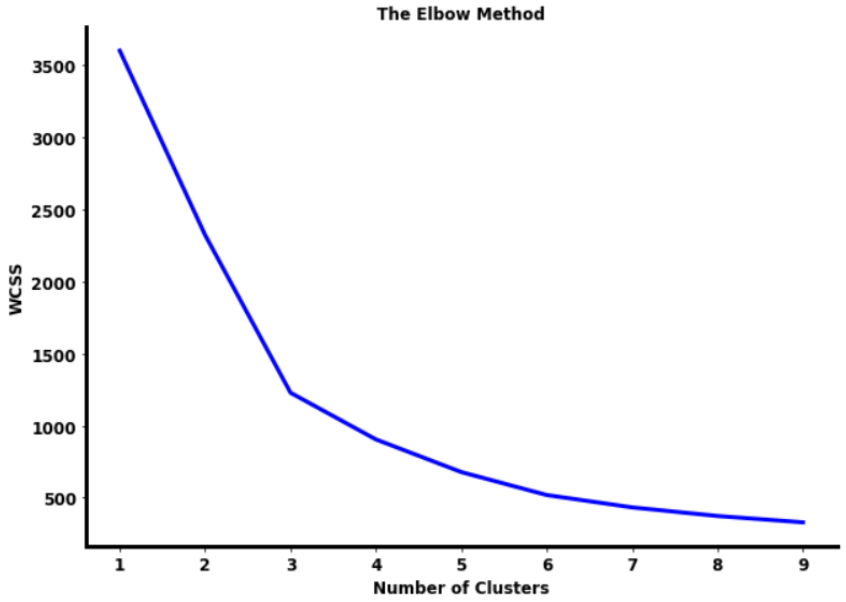


fig3.1.2

Advantages of K-means Clustering:

- ➔ Simple to Implement.
- ➔ Scales to large Dataset.
- ➔ An instance moves to another cluster easily if the centroid value is re computered.
- ➔ Produces tighter clusters than hierarchical clustering.

Disadvantages of K-means Clustering:

- ➔ Manually need to set value for k
- ➔ Initial seeds fed into the model have a large impact on the results of clusters.
- ➔ The order in which data is fed to the model has a significant effect on the results.
- ➔ Sensitive to Scaling.

3.1.1. Implementation of K-means Clustering:

- ➔ Create a BOW Model as shown below

```
]  #Transformation to bag of words
   from sklearn.feature_extraction.text import CountVectorizer
   BOW= CountVectorizer().fit_transform(x).toarray()
```

Fig 3.1.1.1

➔ Create a K means Model as shown below

```
# Modelling K-Means
def k_means(vector,n):
  from sklearn.cluster import KMeans
  km_model = KMeans(n_clusters=n).fit(vector)
  kmeans_labels = km_model.labels_
  y_predicted = km_model.fit_predict(vector)
  cluster_center=km_model.cluster_centers_
  return kmeans_labels,y_predicted,cluster_center
```

Fig 3.1.1.2

➔ Reduce the Dimensions of the predicted vector by creating a PCA model

```
def pca(vector):
  from sklearn.decomposition import PCA
  pca_trans = PCA(n_components=2).fit_transform(vector)
  PCA_components = pd.DataFrame(data = pca_trans)
  return pca_trans,PCA_components
```

Fig 3.1.1.3

➔ Reduce Dimensions as required in a way as shown below

```
pca_bow,PCA_components_bow=pca(BOW)
pca_tfidf,PCA_components_tfidf=pca(TF_iDF)
pca_lda,PCA_components_lda=pca(lda_output)
```

Fig 3.1.1.4

➔ Pass the values to the k-means model

```
clusters,y_predicted,cluster_center=k_means(PCA_components_bow,5)
```

Fig 3.1.1.5

➔ Plot the Cluster Results as shown in the graph below.
We can see that clusters obtained from the above steps give 5 distinct clusters without much overlapping.

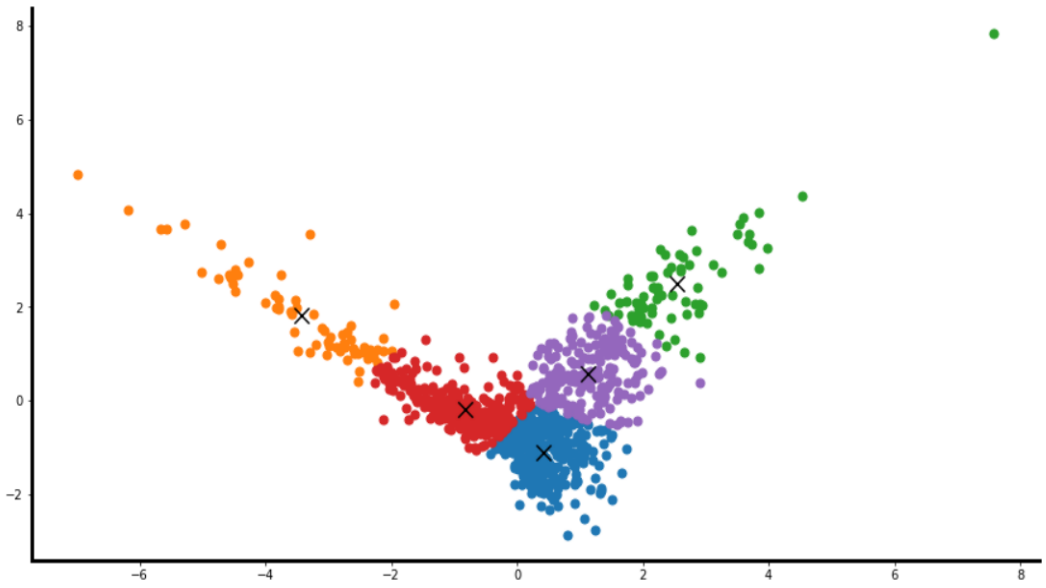


Fig 3.1.1.6

3.1.2. Results of K-means Clustering:

1. **Using Bag Of Words:**
   The graph plotted below is a visualization of the k-means algorithm using BOW as a vectorization technique.
   We can see the algorithm as efficiently predicted 5 clusters without much overlapping of the cluster values.

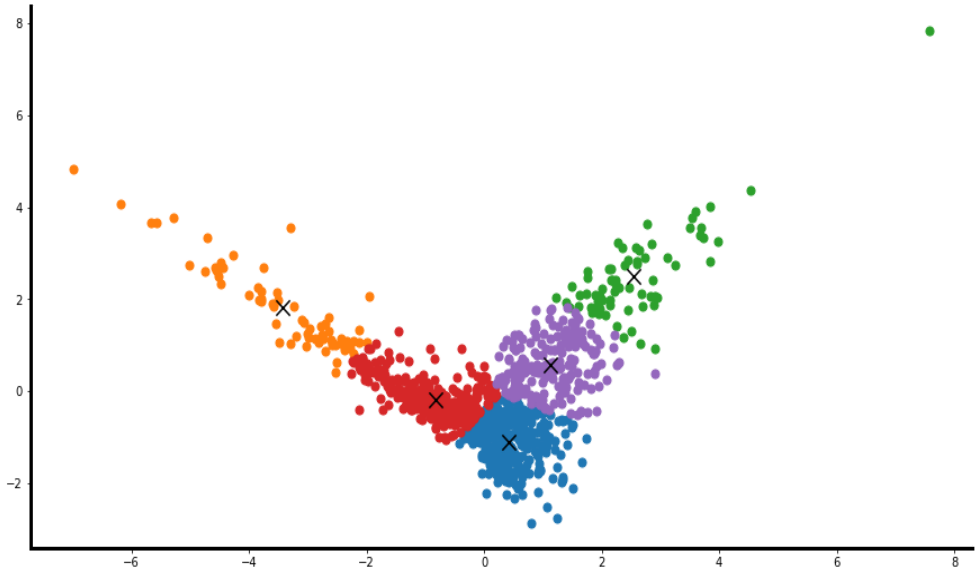


Fig 3.1.2.1

2. **Using TF-iDF:**

   The graph plotted below is a visualization of the k-means algorithm using TF-iDF as a vectorization technique.

   We can see the algorithm as efficiently predicted 5 clusters without much overlapping of the cluster values. We can also observe the cluster results obtained using TF-iDF are much better than BOW.

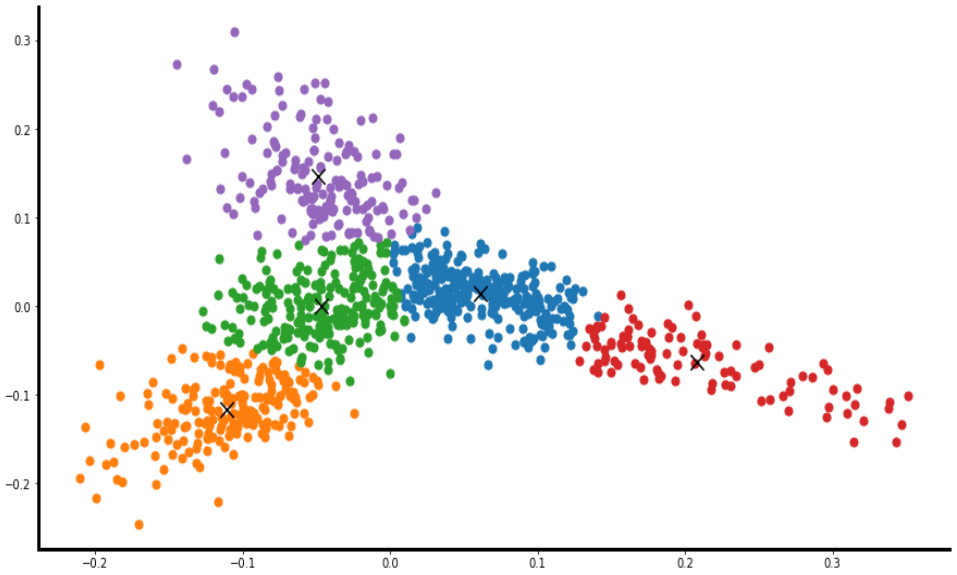


Fig 3.1.2.2

3. **Using LDA:**

   The graph plotted below is a visualization of the k-means algorithm using LDA as a vectorization technique.

   We can see the algorithm as efficiently predicted 5 clusters without much overlapping of the cluster value.

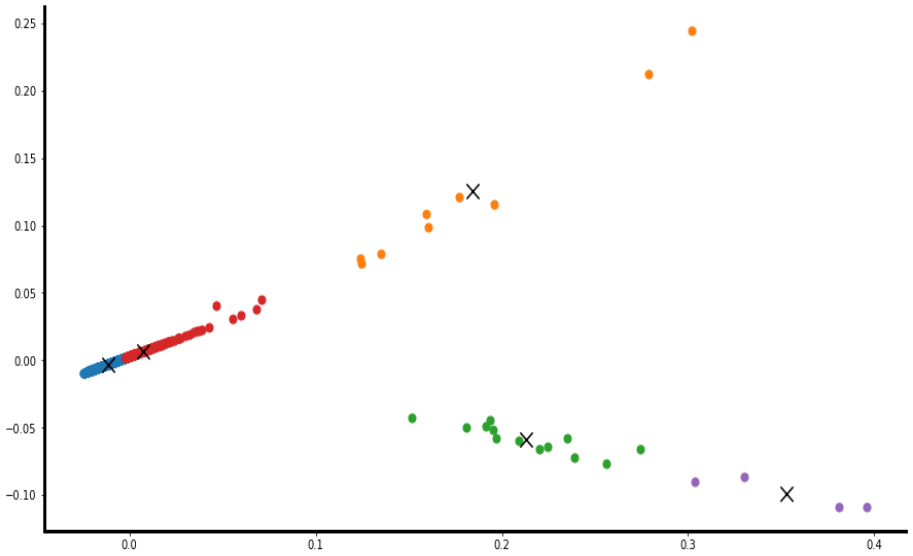


Fig 3.1.2.4

4. **<u>Using Word Embedding:</u>**

   The graph plotted below is a visualization of the k-means algorithm using Word Embedding(Word2Vec)(CBOW)as a vectorization technique.
   We can see the algorithm as efficiently predicted 5 clusters without much overlapping of the cluster values.

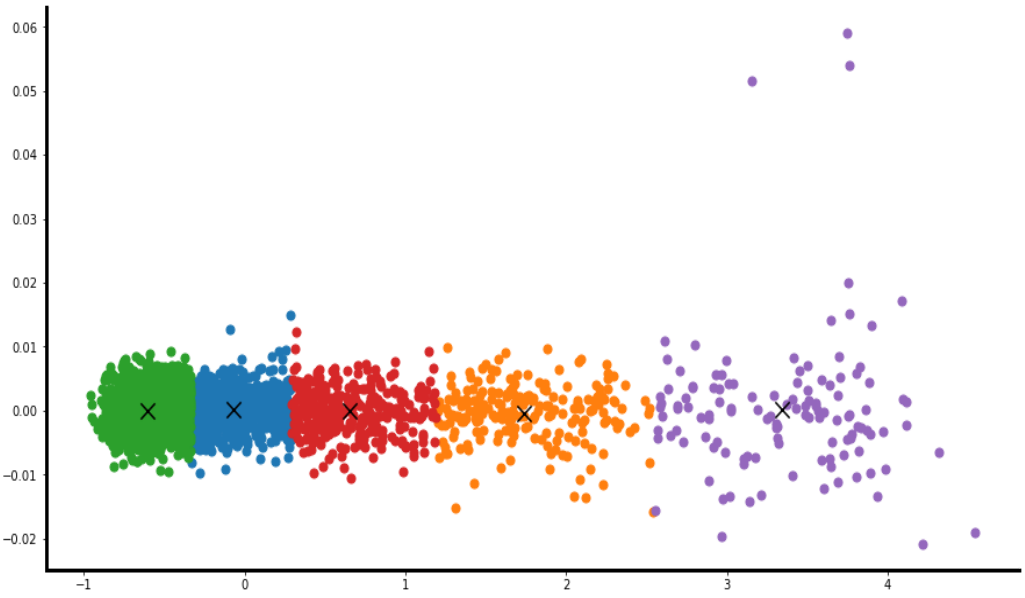


Fig 3.1.2.5

K-Means Clustering using Skip Gram can be seen in the graph plotted below

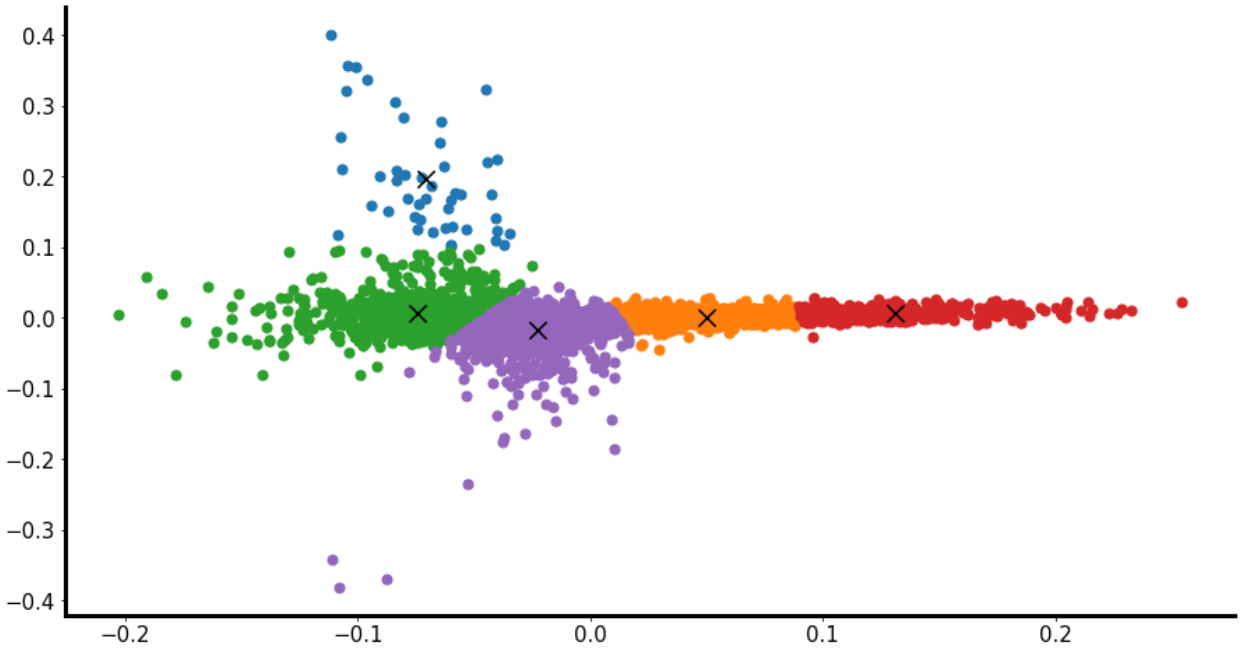


Fig 3.1.2.6

3.2. Hierarchical clustering

This technique is divided into two clustering types:

1. Divisive Hierarchical Clustering
2. Agglomerative Hierarchical Clustering

3.2.1. Divisive Hierarchical Clustering:

We start with all data points as one big cluster and then we iterate and divide these into smaller clusters until the point they can be considered as individual clusters, and we end up with `n` number of small clusters.

3.2.2. Agglomerative Hierarchical Clustering:

This technique is exactly the opposite of the divisive clustering technique. In this technique, each data point is considered as an individual cluster, then these individual clusters merge to form bigger clusters until a point we form one big cluster or `n` number of clusters.

The real-life usage of divisive hierarchical clustering is very limited. Therefore, we have agglomerative Hierarchical Clustering in our code.

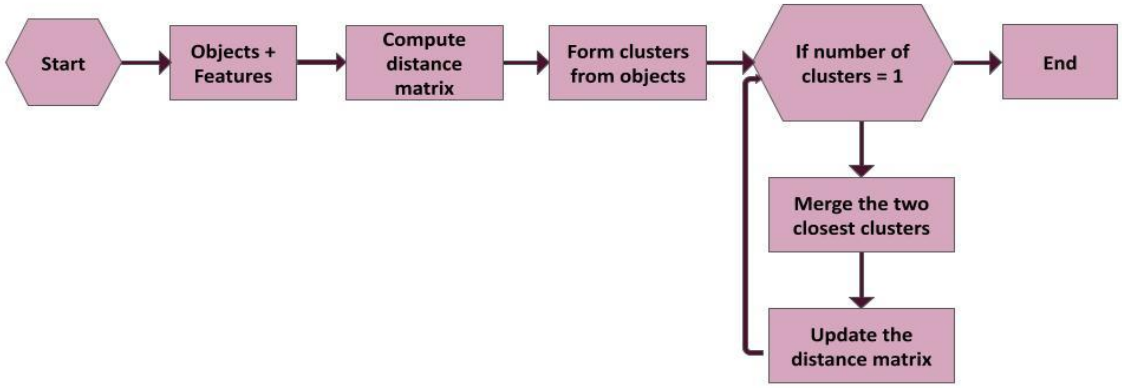


Fig 3.2.2.1

The above diagram explains the flow of agglomerative hierarchical clustering processing technique. We first start with objects and their features. We then create a matrix using dimension reduction techniques such as PCA, cosine similarity, etc. Now we form clusters from the results we get from previous steps. If not the number of clusters is 1, we merge the two closest clusters and update the distance matrix. We continue doing this process repeatedly for clusters of more than 1 until we reach a point where we no longer feel the need to merge more clusters together.

Dendrogram:

This clustering technique uses a dendrogram which is a graph that shows hierarchical relationships between objects. It is mainly used to decide on the number of clusters to pass to the hierarchical algorithm.
We first plot a dendrogram to find out the number of clusters. To find the value of n add a horizontal line across the dendrogram that cuts the long vertical lines and not the small vertical lines(clusters). Now count the total vertical lines that this horizontal line cuts to find the number of clusters to pass to your model.

Advantages of Hierarchical clustering:

- ➔ It shows a hierarchy that is very informative therefore, it is easier to analyze the value of the clusters from a dendrogram.
- ➔ This technique is very easy to understand.

Disadvantages of Hierarchical clustering:

- ➔ It has high space and time complexity therefore, hierarchical clustering can not be used with large amounts of data.
- ➔ Dendrograms can be commonly misinterpreted.
- ➔ It rarely gives good clusters with mixed data-types.

3.2.3. Implementation of Agglomerative Hierarchical Clustering:

Below is the code for implementation of Agglomerative Hierarchical Clustering using bag of Words.

- ➔ Convert the text into vectors using Bag of Words technique as shown below

```
from sklearn.feature_extraction.text import CountVectorizer
BOW= CountVectorizer().fit_transform(x).todense()
```

Fig 3.2.3.1

- ➔ Convert the results into two-dimensional data using Principal Component Analysis as shown below

```
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
pca = PCA(n_components=2)
pca_bow = pca.fit_transform(BOW)
tsne_bow=TSNE(n_components=2).fit_transform(BOW)
# Plot the explained variances
features = range(pca.n_components_)
plt.bar(features, pca.explained_variance_ratio_, color='cyan')
plt.xlabel('PCA features')
plt.ylabel('variance %')
plt.xticks(features)
# Save components to a DataFrame
PCA_components = pd.DataFrame(data = pca_bow)
TSNE_components=pd.DataFrame(data=tsne_bow)
print(BOW.shape)
print(PCA_components.shape)
print(PCA_components)
```

Fig 3.2.3.2

➔ The two-dimensional results obtained after PCA can be used to pass as an attribute to form a dendrogram as shown below.

```
linkage_matrix = ward(PCA_components)
dendo = plt.subplots(figsize=(15, 10))
dendo = dendrogram(linkage_matrix, orientation="top");
plt.axhline(y=20, color='r', linestyle='--')
plt.title('Agglomerative Clustering-BOW', fontsize = 12,fontweight='bold')
plt.tight_layout()
```

Fig 3.2.3.3

➔ The number of clusters obtained from the dendrogram can be used now to pass to our agglomerative hierarchical clustering model.

```
# Implementing the hierarchical clustering.
hc = AgglomerativeClustering(n_clusters=5, linkage='ward')
hc_predict = hc.fit_predict(PCA_components)
```

Fig 3.2.3.4

➔ Now, we visualise the results on a scatter plot as shown below.

```
#Now plot the results in a scatterplot.
for i in range(len(hc_predict)):
    filtered_label_hc = PCA_components[hc_predict == i]
    plt.scatter(filtered_label_hc.iloc[:,0] , filtered_label_hc.iloc[:,1] , s = 30)
fig = plt.gcf(); fig.set_size_inches(15, 8)
plt.title('Agglomerative Clustering-BOW', fontsize = 12,fontweight='bold')
plt.xlabel("PC1", fontsize = 12,fontweight='bold')
plt.ylabel("PC2", fontsize = 12,fontweight='bold')
import matplotlib as mpl
mpl.rcParams["axes.spines.right"] = False; mpl.rcParams["axes.spines.top"] = False
from pylab import *
rc('axes', linewidth=3)
fontsize = 12
ax = gca()
for tick in ax.xaxis.get_major_ticks():
    tick.label1.set_fontsize(fontsize)
    tick.label1.set_fontweight('bold')
for tick in ax.yaxis.get_major_ticks():
    tick.label1.set_fontsize(fontsize)
    tick.label1.set_fontweight('bold')
#plt.figure()
```

Fig 3.2.3.5

Similarly, we write code for other feature engineering techniques as well.

3.2.3. Results of Agglomerative Hierarchical Clustering:

1. **Using Bag Of Words:**

Dendrogram:

The horizontal line displayed in the dendrogram below cuts the clusters at five different points which means it is safe to say that the number of clusters can be assumed as five.

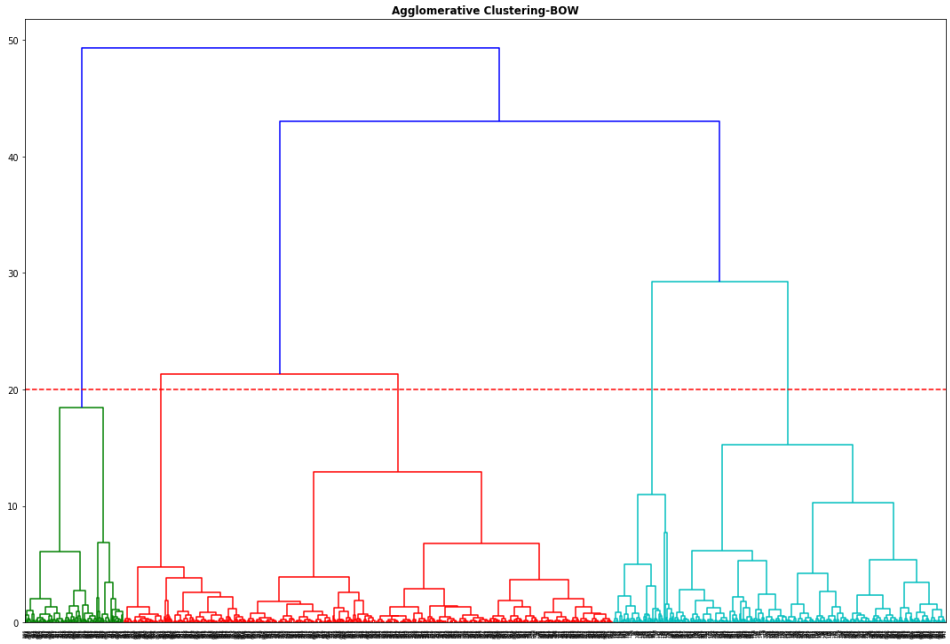


Fig 3.2.3.1

Scatter Plot:

The below visualisation displays scatter plots with five different clusters. Each cluster is scattered separately and there isn't a lot of overlapping.

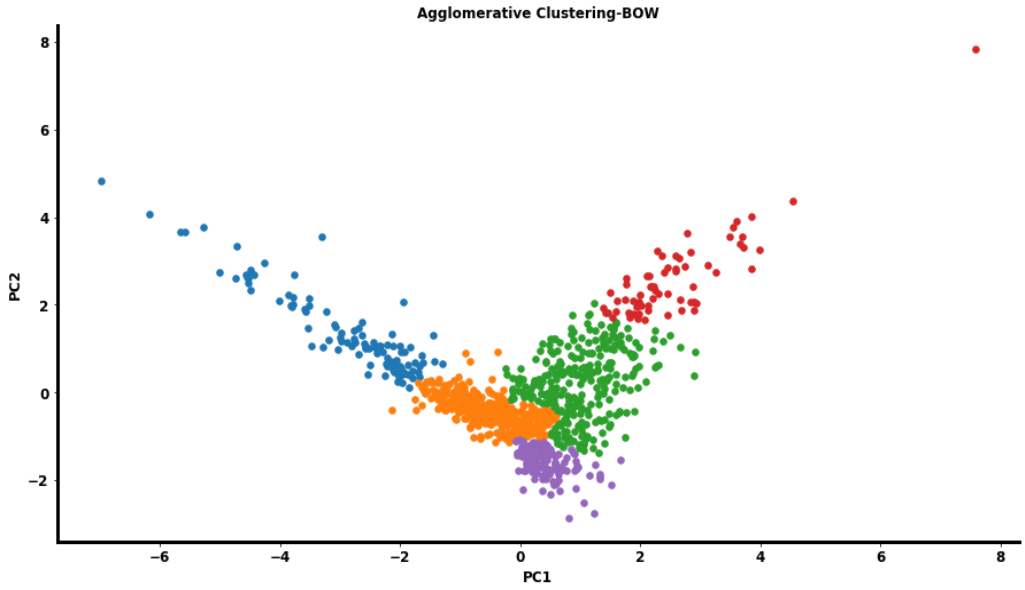


Fig 3.2.3.2

2. **Using TF-iDF:**

Dendrogram:

The horizontal line displayed in the dendrogram below cuts the clusters at five different points which means it is safe to say that the number of clusters can be assumed as five.

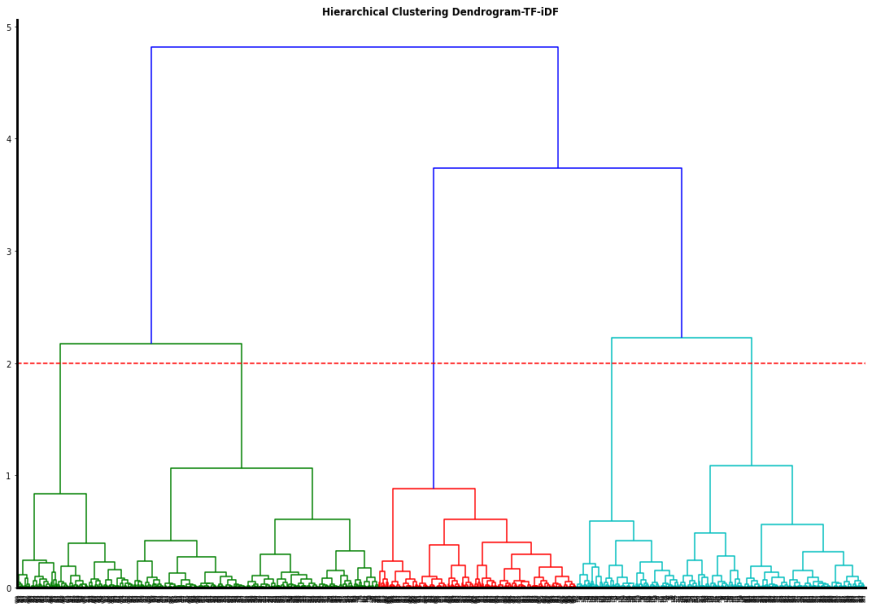


Fig 3.2.3.3

Scatter Plot: The below visualisation displays scatter plots with five different clusters. Each cluster is scattered separately, and there is minimal overlapping over each other.

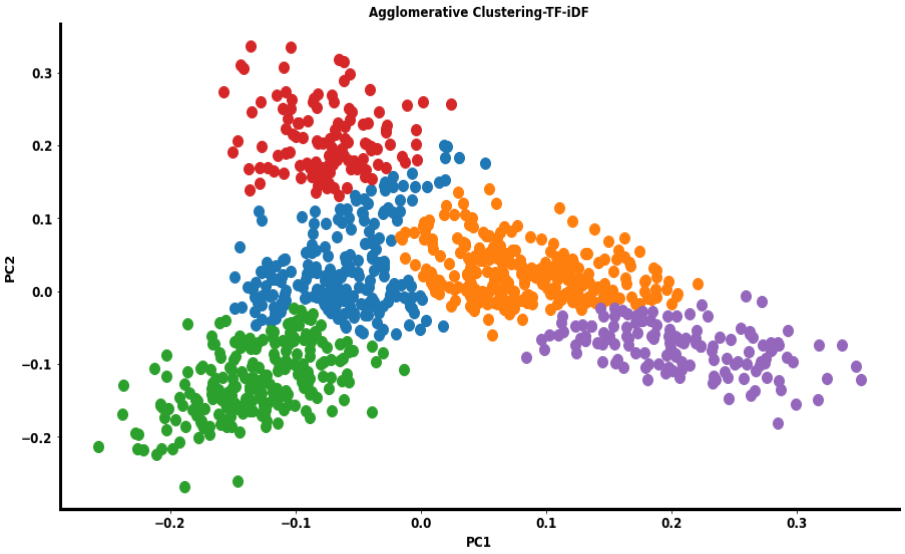


Fig 3.2.3.4

3. **Using LDA:**

Dendrogram:

The horizontal line displayed in the dendrogram below cuts the clusters at five different points which means it is safe to say that the number of clusters can be assumed as five.

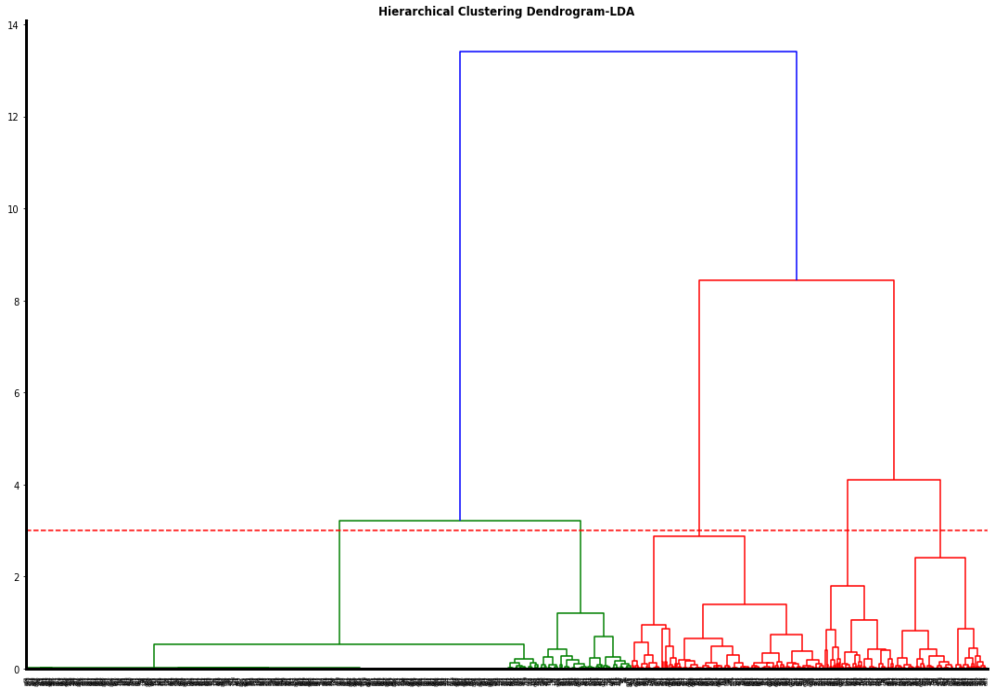


Fig 3.2.3.5

Scatter Plot:

The below visualisation displays scatter plots with five different clusters. Each cluster is scattered separately but the structure of clusters doesn't seem very appropriate.

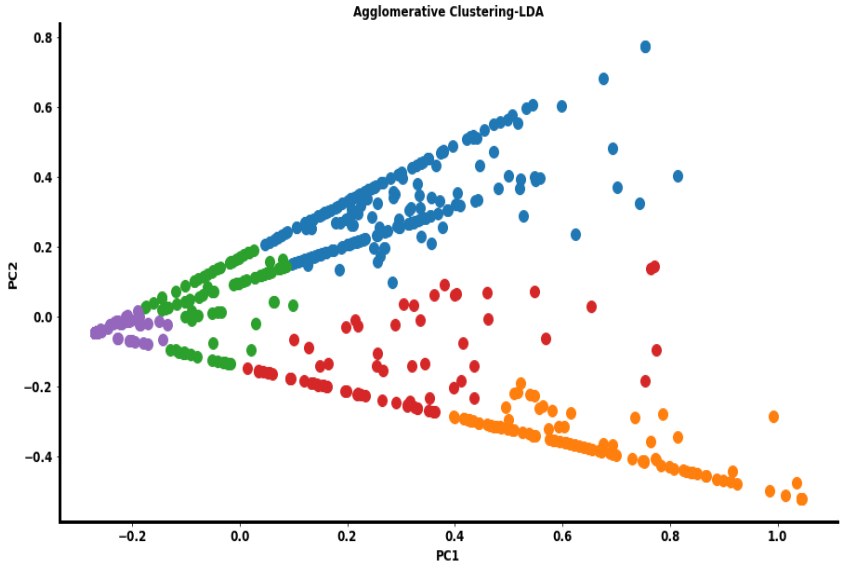


Fig 3.2.3.6

4. **Using Word Embedding:**

Dendrogram:

The horizontal line displayed in the dendrogram below cuts the clusters at five different points which means it is safe to say that the number of clusters can be assumed as five.

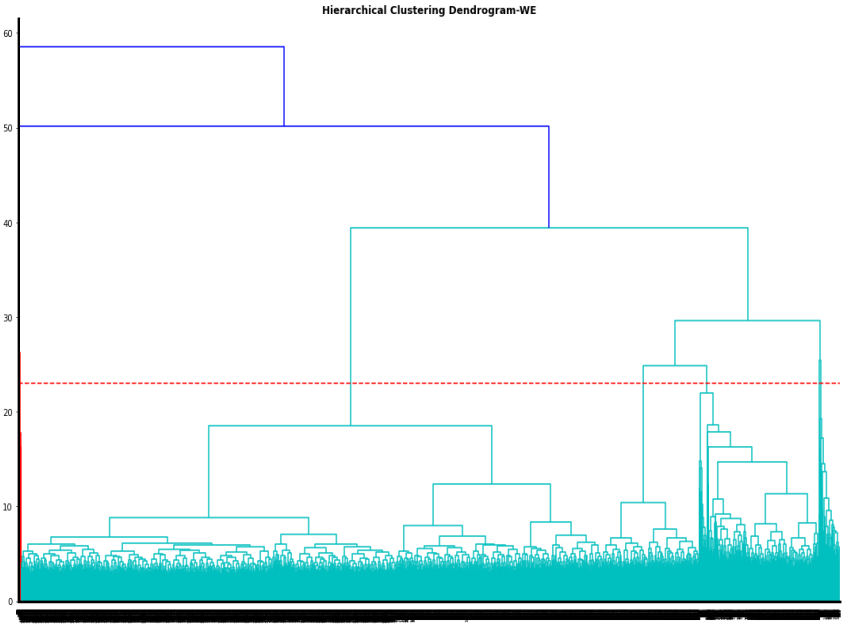


Fig 3.2.3.7

Scatter Plot:

The below visualisation displays scatter plots with five different clusters. Each cluster is scattered separately and the results produced seem very uniform with some overlapping of clusters.

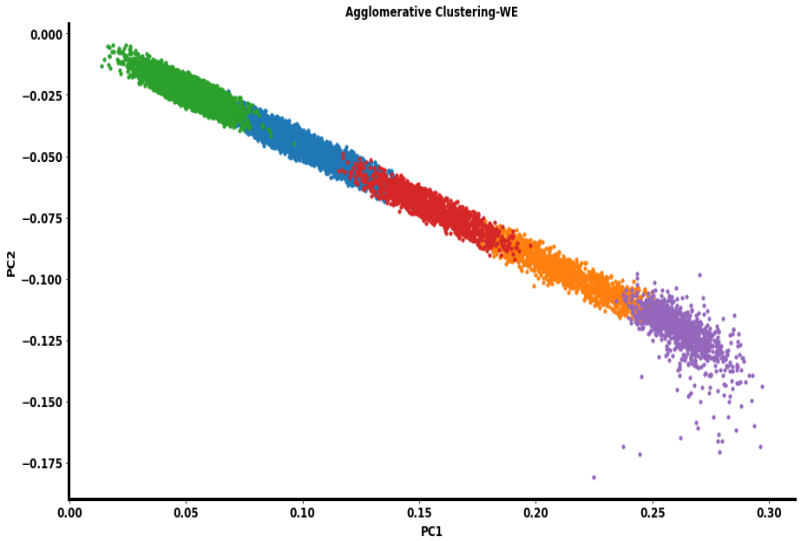


Fig 3.2.3.8

3.3. Expectation Maximization Clustering

A general technique for finding maximum likelihood estimators in latent variable models is the expectation-maximization (EM) algorithm.
EM clustering algorithm is an approach for finding maximum likelihood between latent Variables.EM work is a two-step clustering approach:

Step1: E-Step: Estimate the missing values in the data set.
Step2: M-Step: Maximize the parameters of the model in the presence of the data.

Advantages of Expectation Maximization Clustering :

- ➔ Gives useful results for real-world Data sets.
- ➔ EM performs faster if analytical expressions are available for M steps
- ➔ Derivative Free Optimiser

Disadvantages of Expectation Maximization Clustering:

- ➔ High Complexity.
- ➔ Requires both forward and backward probabilities.
- ➔ Hessian needs to be calculated manually.

3.3.1. Implementation of Expectation Maximization Clustering:

- ➔ Create an EM model using GaussianMixture() from sklearn library as shown in the figure below.

```
#Expectation Maximization
def EM(vector):
    from sklearn.mixture import GaussianMixture
    #from sklearn import preprocessoring
    gmm = GaussianMixture(n_components=5, random_state=0).fit(vector)
    EM_clusters = gmm.predict(vector)
    return EM_clusters
```

Fig 3.3.1

- ➔ Train the above generated model with TF-iDF input features as shown below.

```
# Train the model with obtained features
EM_clusters = EM(TF_iDF)
```

Fig 3.3.2

- ➔ Reduce the dimensions of the above trained model using TSNE to produce better insights of the clustering results as shown below.

```
def cluster_visualization(vector,cluster_model,title):
  # title=[title1, title2]
  from sklearn.decomposition import PCA
  PCA_points = PCA(n_components=2).fit_transform(vector)
  fig, ax = plt.subplots(figsize=(16,8))
  ax.scatter(PCA_points[:, 0], PCA_points[:, 1], c=cluster_model, s=40, cmap='viridis')
  ax.set_title(title[0], fontsize = 12,fontweight='bold')
  ax.set_xlabel('PC1', fontsize = 12,fontweight='bold')
  ax.set_ylabel('PC2', fontsize = 12,fontweight='bold')
  from sklearn.manifold import TSNE
  TSNE_points = TSNE(n_components=2).fit_transform(vector) # Fit and transform
  fig, ax = plt.subplots(figsize=(16,8))
  ax.scatter(TSNE_points[:, 0], TSNE_points[:, 1], c=cluster_model, s=40, cmap='viridis')
  ax.set_title(title[1], fontsize = 12,fontweight='bold')
  ax.set_xlabel('PC1', fontsize = 12,fontweight='bold')
  ax.set_ylabel('PC2', fontsize = 12,fontweight='bold')
```

Fig 3.3.3

➔ Again we reduce the dimensions of the trained matrix using PCA as shown below.

```
import matplotlib as mpl
from pylab import *
def cluster_visualization1(vector,cluster_model,title):
  # title=[title1, title2]
  from sklearn.decomposition import PCA
  PCA_points = PCA(n_components=2).fit_transform(vector)
  fig, ax = plt.subplots(figsize=(16,8))
  ax.scatter(PCA_points[:, 0], PCA_points[:, 1], c=cluster_model, s=40, cmap='viridis')
  ax.set_title(title[0], fontsize = 12,fontweight='bold')
  ax.set_xlabel('PC1', fontsize = 12,fontweight='bold')
  ax.set_ylabel('PC2', fontsize = 12,fontweight='bold')


  mpl.rcParams["axes.spines.right"] = False; mpl.rcParams["axes.spines.top"] = False
  rc('axes', linewidth=3)
  fontsize = 12
  ax1 = gca()
  for tick in ax1.xaxis.get_major_ticks():
    tick.label1.set_fontsize(fontsize)
    tick.label1.set_fontweight('bold')
  for tick in ax1.yaxis.get_major_ticks():
    tick.label1.set_fontsize(fontsize)
    tick.label1.set_fontweight('bold')

  from sklearn.manifold import TSNE
  TSNE_points = TSNE(n_components=2).fit_transform(vector) # Fit and transform
  fig, ax = plt.subplots(figsize=(16,8))
  ax.scatter(TSNE_points[:, 0], TSNE_points[:, 1], c=cluster_model, s=40, cmap='viridis')
  ax.set_title(title[1], fontsize = 12,fontweight='bold')
  ax.set_xlabel('PC1', fontsize = 12,fontweight='bold')
```

```
  ax.set_ylabel('PC2', fontsize = 12,fontweight='bold')

  mpl.rcParams["axes.spines.right"] = False; mpl.rcParams["axes.spines.top"] = False
  rc('axes', linewidth=3)
  fontsize = 12
  ax1 = gca()
  for tick in ax1.xaxis.get_major_ticks():
    tick.label1.set_fontsize(fontsize)
    tick.label1.set_fontweight('bold')
  for tick in ax1.yaxis.get_major_ticks():
    tick.label1.set_fontsize(fontsize)
    tick.label1.set_fontweight('bold')
```

Fig 3.3.4

➔ Finally, we try to visualize the results for the EM model as shown below.

```
title=['EM-TF-iDF Clustering using PCA','EM-TF-iDF  Clustering using TSNE']
EM_TFiDF_visualization=cluster_visualization(TF_iDF,EM_clusters,title)
```

Fig 3.3.5

3.3.2. Results of Expectation Maximization Clustering:

1. **Using Bag Of Words:**

EM-BOW using PCA:

The graph shown below is the clustering result for the EM model using BOW feature Engineering technique. We have implemented it using the PCA model and we can observe that there are 5 different clusters where it appears that clusters 2 and 3 overlap each to some extent.

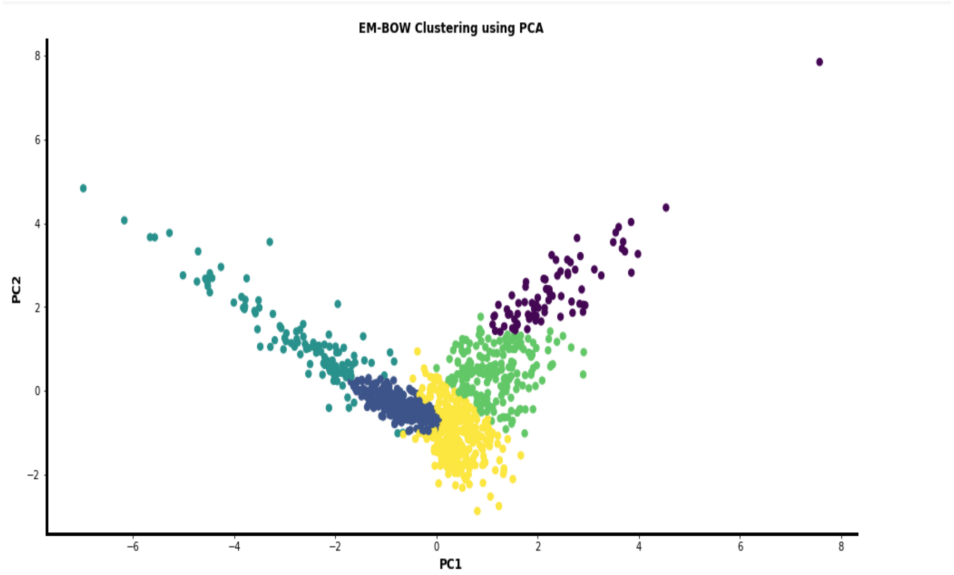


Fig 3.3.2.1

2. **Using TF-iDF:**
   The graph shown below is the clustering result for the EM model using TF-iDf technique.
   We have implemented it using the PCA model and we can observe that there are 5 different clusters where it appears that all five slightly overlap with each other.

EM-TF-iDF using PCA:

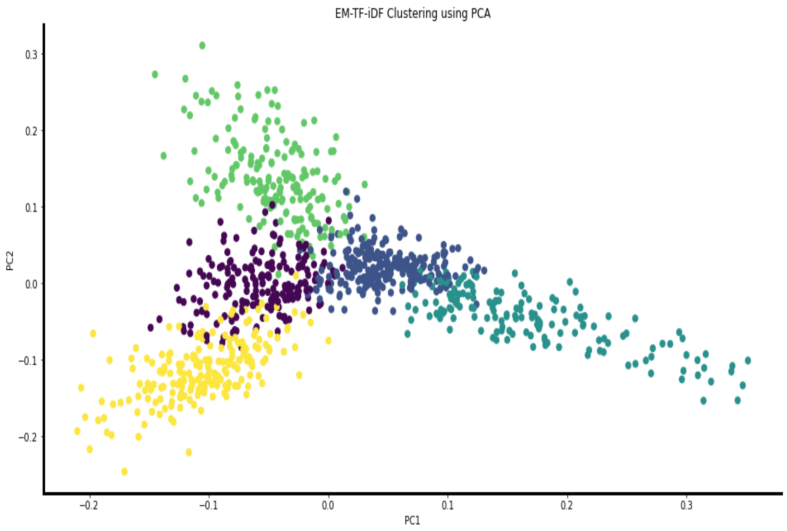


Fig 3.3.2.2

3. **Using LDA:**

EM-LDA using PCA:

The graph shown below is the clustering result for the EM model using LDA feature Engineering technique. The results look dissatisfactory as the clusters are not very aligned.

It is implemented using the PCA model.

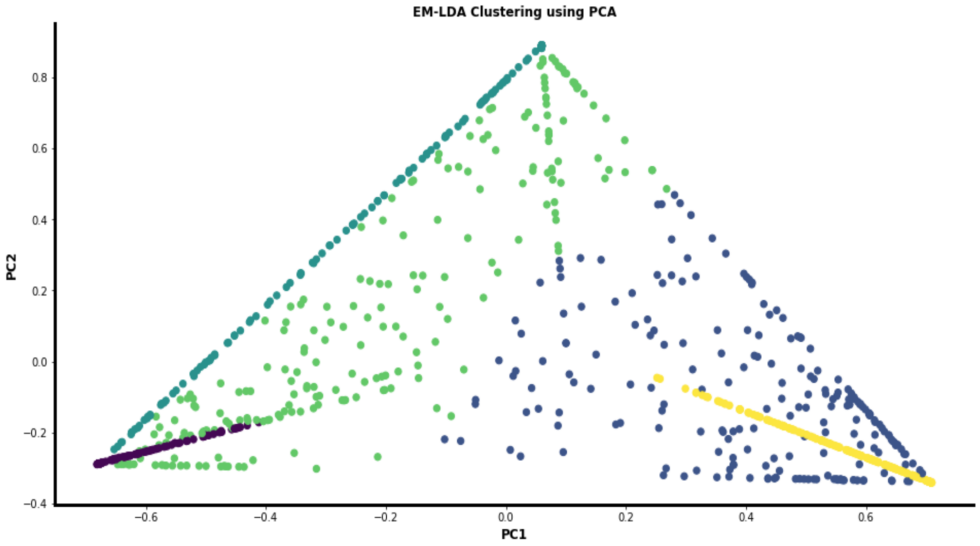


Fig 3.3.2.3

4. **Using Word Embedding:**

EM-WE using PCA:

The graph shown below is the clustering result for the EM model using Word Embedding technique.
We have implemented it using the PCA model.

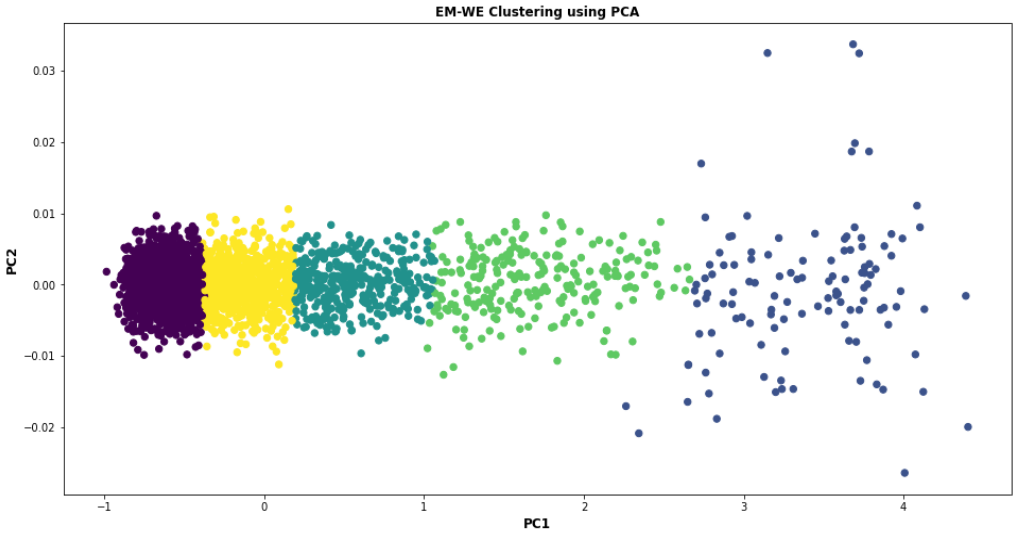


Fig 3.3.2.4

4. Best Model Selection

At this point, we have selected the **K-means clustering algorithm** using **Word Embedding** as our preferred model to do the predictions. We have chosen the following Model based on the error analysis and performance of the model compared with other models based on Silhouette Coefficient.

➔ Silhouette score for all the clustering model can be seen in the graph below:

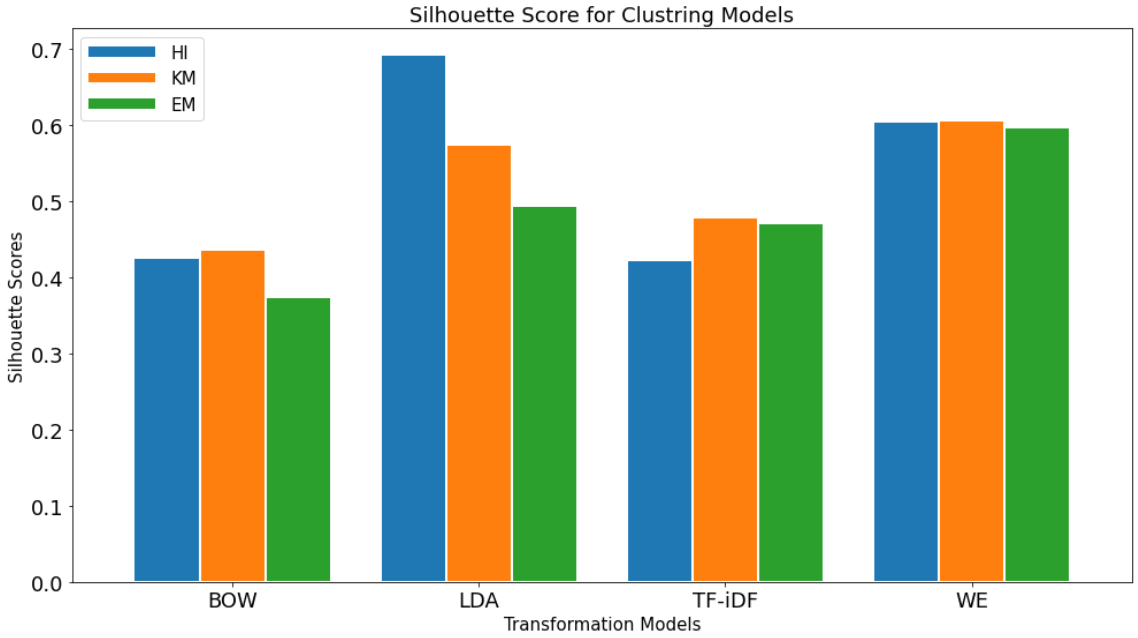


fig4.1

➔ From the above graph, we can observe that the silhouette score for the K-means model is quite consistent for all four featuring models.

- ➔ Even though Hierarchical clustering performs better with LDA but K-means gives better results with the remaining three techniques.
- ➔ Also, we can observe that the word embedding featuring techniques gives a very consistent silhouette score for all the models.
- ➔ We can see that the score is almost between (0.5-0.6) which depicts that the clusters formed are quite separate from each other, which is better than other modelling techniques.

The same details can be inferred from the table below as well. It depicts the silhouette score of all three clustering techniques.

	HI	KM	EM
BOW	0.426105	0.435329	0.373399
LDA	0.692035	0.573739	0.493751
TF-iDF	0.422048	0.478122	0.470143
WE	0.604522	0.606443	0.597140

Fig 4.2

Silhouette coefficient for Hierarchical clustering model

Silhouette score for Hierarchical clustering seems to be the highest for LDA followed by word embedding.

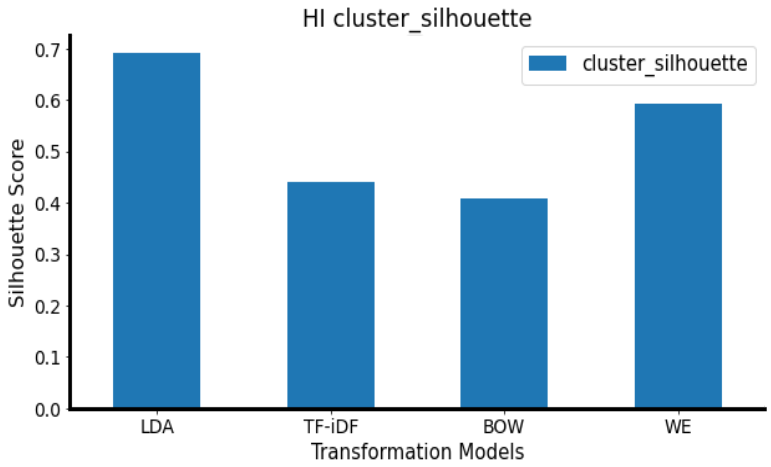


fig4.3

Silhouette coefficient for K-Means model

Silhouette score for K-means clustering seems to be the highest for word embedding followed closely by other techniques.

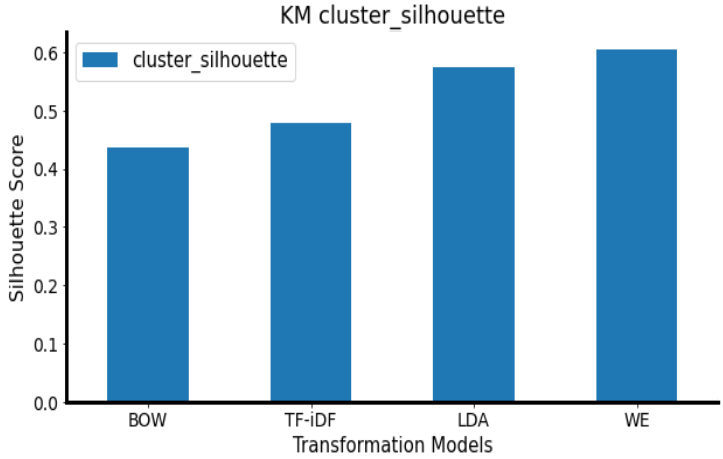


Fig 4.3

Silhouette coefficient for EM model

Silhouette score for the EM clustering model seems to be the highest for word embedding followed by LDA.

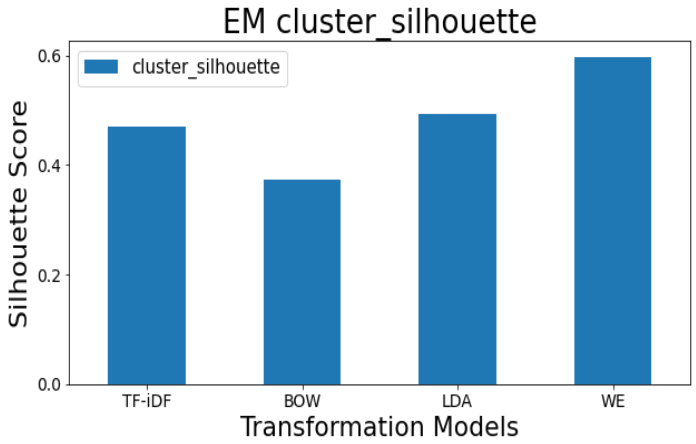


Fig 4.4

5. Error Analysis

➔ **Coherence Analysis**

Coherence means the connection or consistency between two things. Coherence analysis is used to check the semantic similarity between the words in a document. It is majorly used to evaluate topic modelling.
We can figure out the number of topics to pass to our LDA model by calculating coherence values for multiple numbers and the one which gives the highest value can be chosen as the ideal number of topics for our model. This is useful in error analysis as it helps to identify the correct number of clusters to pass to the LDA model. Higher coherence number is considered to be better.

Code for Coherence Analysis:

```
from gensim.models.coherencemodel import CoherenceModel
coherence_model_lda = CoherenceModel(model=lda_model, texts=dataset, dictionary=dictionary , coherence='c_v')
coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score: ', coherence_lda)
```

Fig 5.1

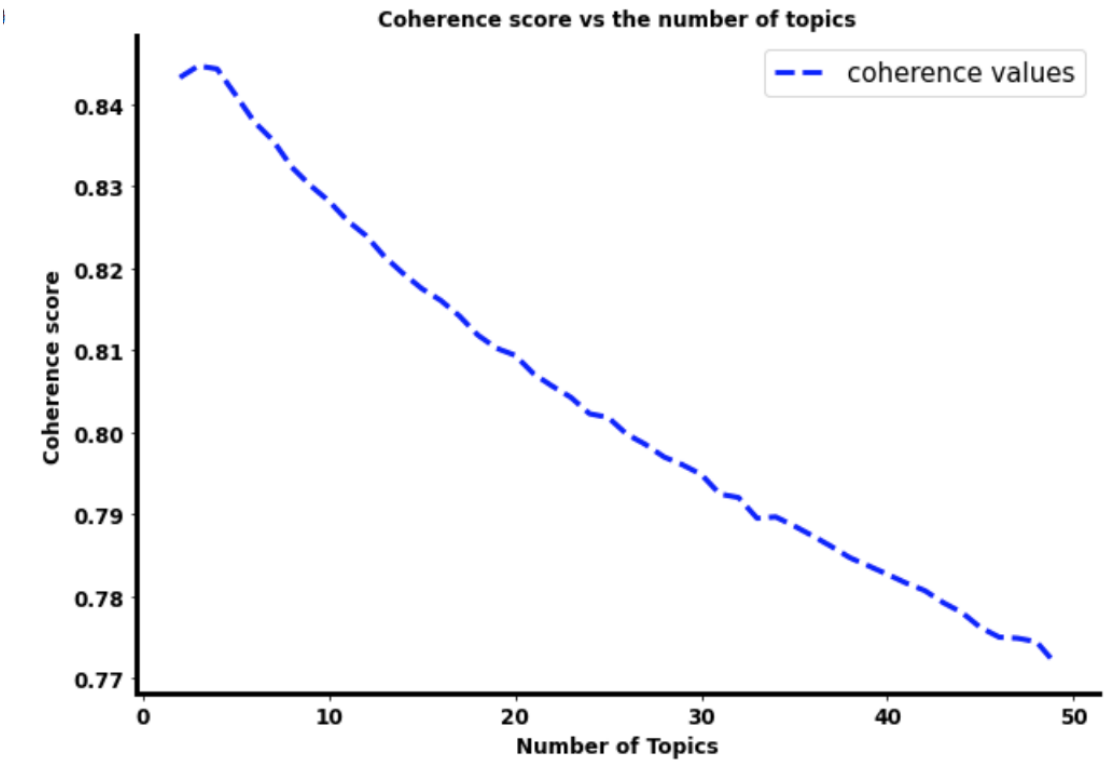

Coherence score vs the number of topics
coherence values
Coherence score
0.84
0.83
0.82
0.81
0.80
0.79
0.78
0.77
0
10
20
30
40
50
Number of Topics

fig5.2

```
# The coherence scores for multiple number of topics.
for m, cv in zip(x, coherence_values):
    print("Num Topics =", m, " has Coherence Value of", round(cv, 4))
```

```
Num Topics = 2  has Coherence Value of 0.8433
Num Topics = 3  has Coherence Value of 0.8447
Num Topics = 4  has Coherence Value of 0.8443
Num Topics = 5  has Coherence Value of 0.8411
Num Topics = 6  has Coherence Value of 0.8378
Num Topics = 7  has Coherence Value of 0.8356
Num Topics = 8  has Coherence Value of 0.8324
Num Topics = 9  has Coherence Value of 0.8301
Num Topics = 10  has Coherence Value of 0.8282
Num Topics = 11  has Coherence Value of 0.8258
Num Topics = 12  has Coherence Value of 0.824
Num Topics = 13  has Coherence Value of 0.8214
Num Topics = 14  has Coherence Value of 0.8193
Num Topics = 15  has Coherence Value of 0.8175
Num Topics = 16  has Coherence Value of 0.8161
Num Topics = 17  has Coherence Value of 0.8142
Num Topics = 18  has Coherence Value of 0.8119
Num Topics = 19  has Coherence Value of 0.8103
Num Topics = 20  has Coherence Value of 0.8094
Num Topics = 21  has Coherence Value of 0.8071
Num Topics = 22  has Coherence Value of 0.8056
Num Topics = 23  has Coherence Value of 0.8043
Num Topics = 24  has Coherence Value of 0.8023
Num Topics = 25  has Coherence Value of 0.8018
```

Fig 5.3

➔ **Kappa Coefficient**

Cohen's Kappa statistic is a very useful metric. Kappa coefficient is used to calculate how closely the classification done by our machine learning algorithms relates to the labelled data. The value is always equal to or less than 1. If the value is less than 0, it means that the model is useless. Higher the value, the better the model is.

The graph below depicts the kappa coefficient for all three clustering techniques.

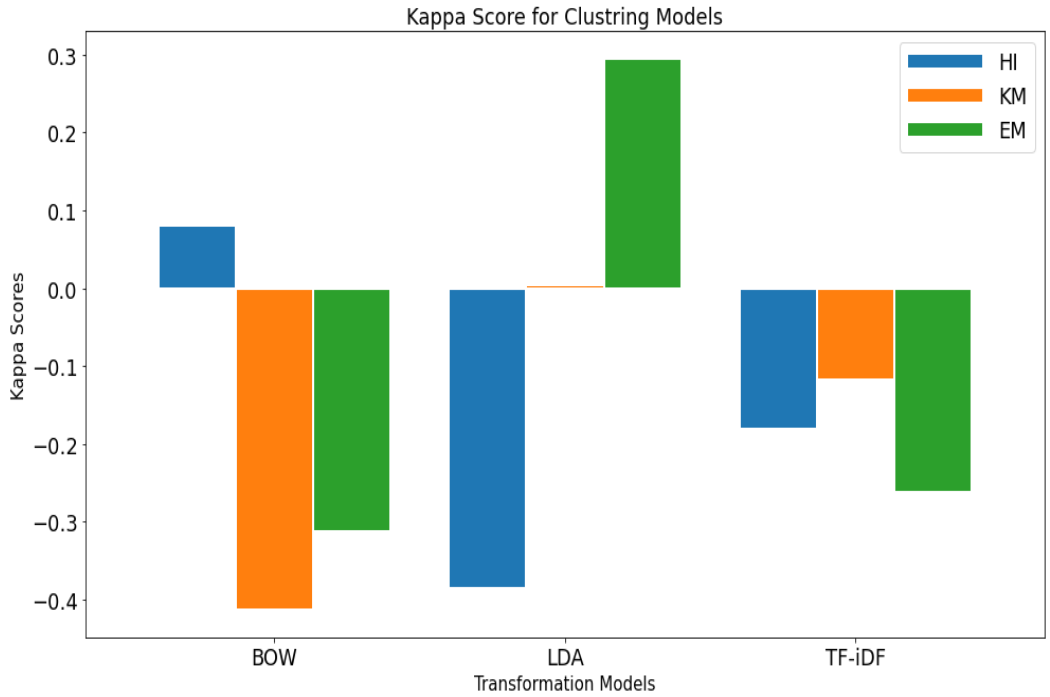


fig5.4

➔ From the above graph we can observe that the kappa coefficient is negative for all the clustering models using TF-iDF. Negative kappa means that the model is not usable. Therefore, it helps in error analysis.

➔ **Silhouette Coefficient**

Silhouette score or coefficient is used to measure how good a clustering model is and estimates the average distance between the clusters. Its range varies from -1 to 1.

- Range = 1 means that the clusters are apart from each other.
- Range = 0 means that the clusters and not different
- Range = -1 means that the clusters are assigned incorrectly.

➔ **Completeness:**

A perfectly complete clustering is one where all data-points belonging to the same class are clustered into the same cluster. Completeness describes the closeness of the clustering algorithm to this perfection.

➔ **Homogeneity**:

This parameter is useful to check whether the sample belongs to the correct cluster. Its value should be between 0 and 1 where a lower value indicates low homogeneity.

Silhouette Coefficient can also be used as a part of error analysis apart from helping us with choosing the champion model.
In the example below, we have visualised silhouette analysis for k-means algorithm. We notice that we see negative values for cluster = 3 and cluster = 1. This means that using clusters = 4 would not produce good results. To choose the ideal number of clusters, we should make sure that there is no negative value displaying for any of the clusters.

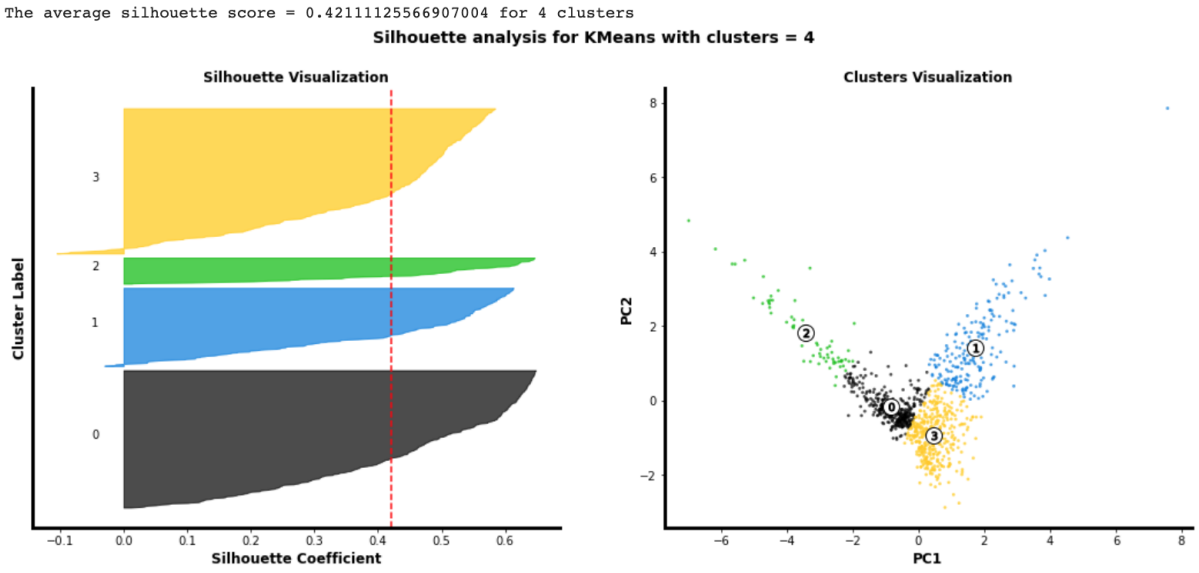


Fig 5.5

Code for Clustering Metrics:

```
# Clustering Mertrics calculation
def clustering_metrics(labels,clusters,vector,model):
  from scipy.stats import spearmanr
  #from time import time
  from sklearn import metrics
  kappa=metrics.cohen_kappa_score(labels, clusters,weights='linear')
  cluster_silhouette=metrics.silhouette_score(vector, clusters, metric='euclidean'),
  human_silhouette=metrics.silhouette_score(vector,labels,metric='euclidean')
  homogeneity=metrics.homogeneity_score(labels, clusters)
  completeness=metrics.completeness_score(labels, clusters)
  v_meas=metrics.v_measure_score(labels, clusters)
  ARI=metrics.adjusted_rand_score(labels, clusters)
  AMI=metrics.adjusted_mutual_info_score(labels,clusters)
  spearman=spearmanr(labels,clusters)
  cluster_metrics=pd.DataFrame({'Model':model,'kappa':kappa,'cluster_silhouette':cluster_silhouette,
                                'human_silhouette':human_silhouette, 'homogeneity':homogeneity,
                                'completeness':completeness,'v_meas':v_meas,
                               'ARI':ARI,'AMI':AMI,'correlation':spearman[0],'pvalue':spearman[1]})
  return cluster_metrics
```

Fig 5.6

Results of metrics for hierarchical clustering:

The below table shows the error analysis Hierarchical clustering Algorithm, We can see the different parameters that can help us to understand the efficiency of the model. We can see that the silhouette coefficient is highest for the LDA model while it is lowest for the BOW model. Whereas, we can observe poor homogeneity and completeness for the LDA model and good for the Tf-Idf model. Also, we can observe that the kappa coefficient is negative for all the three models which indicates that clustering models have not efficiently clustered the data points.

	kappa	cluster_silhouette	human_silhouette	homogeneity	completeness	v_meas	ARI	AMI	correlation	pvalue
BOW	-0.416387	0.390201	0.116731	0.346398	0.402508	0.372351	0.208833	0.368947	-0.570368	2.31757e-87
TF-iDF	-0.180334	0.422044	0.341363	0.633994	0.654502	0.644085	0.554653	0.642266	-0.148675	2.33664e-06
WE	--	0.576972	--	--	--	--	--	--	--	--
LDA	-0.384894	0.692035	-0.153249	0.259757	0.322028	0.28756	0.209046	0.283576	-0.60956	8.89274e-103

Table 5.7

Visualization for Error Analysis Matrix for Hierarchical Clustering

The below graph displays the results of all the metrics we used to analyse the hierarchical clustering. Overall the results are acceptable but see negative values for kappa coefficient, human silhouette and correlation.

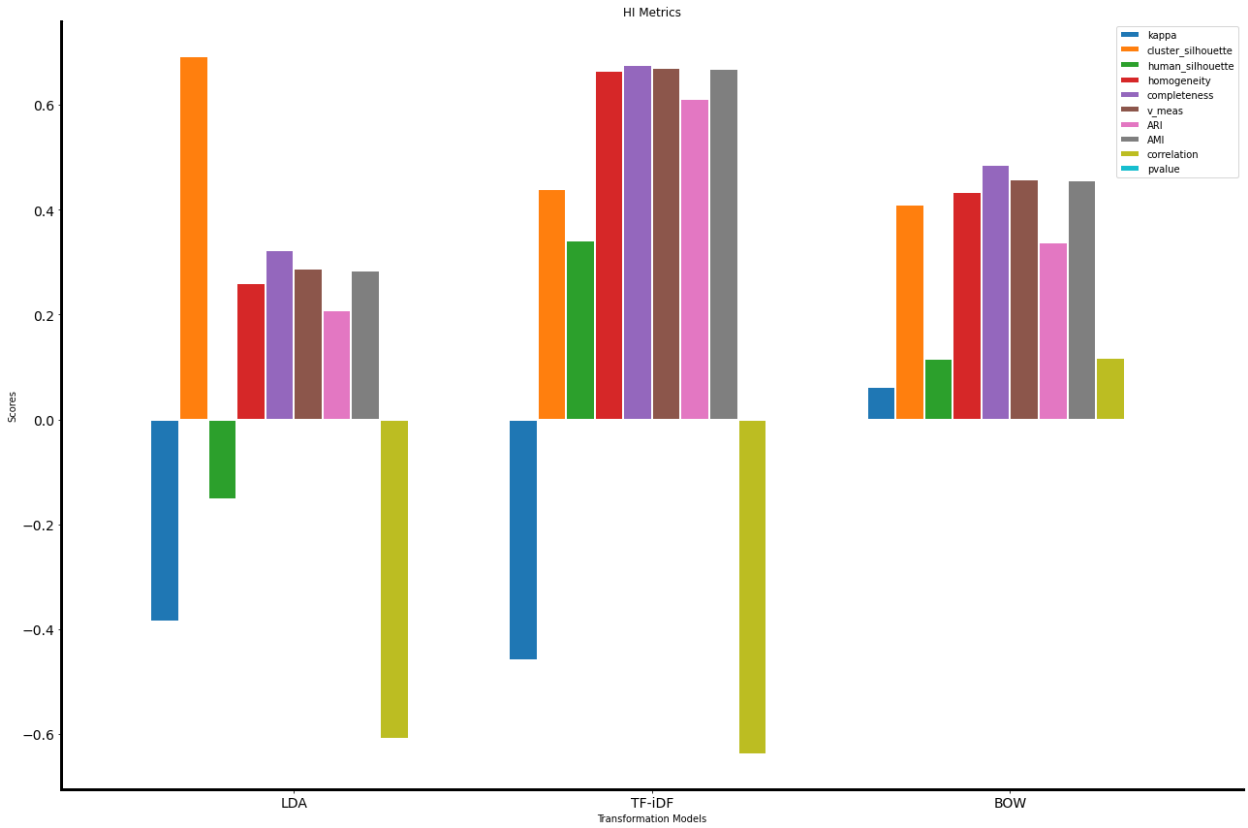


Fig 5.8

Results of Matrix for K-means clustering:

The below table shows the error analysis K-Means clustering Algorithm, We can see the different parameters that can help us to understand the efficiency of the model.
We can see that the silhouette coefficient is highest for Word Embedding model while it is lowest for the BOW model. Whereas, we can observe poor homogeneity and completeness for the BOW model and good for the Tf-Idf model. Also, we can observe that the kappa coefficients are negative for all the three models which indicates that clustering model have not efficiently clusters the data points.

	kappa	cluster_silhouette	human_silhouette	homogeneity	completeness	v_meas	ARI	AMI	correlation	pvalue
BOW	-0.389578	0.435116	0.116713	0.437865	0.495932	0.465093	0.354553	0.462225	-0.548614	1.15682e-79
TF-iDF	-0.125203	0.478003	0.305813	0.656588	0.682516	0.669301	0.592212	0.667605	-0.0735752	0.0199708
LDA	-0.0082678	0.574074	-0.186475	0.0877877	0.191909	0.120468	0.0663051	0.113618	-0.000470583	0.988142
WE	--	0.604488	--	--	--	--	--	--	--	--

Table 5.9

Visualization for Error Analysis Matrix for K-Means

The below graph displays the results of all the metrics we used to analyse the k-means clustering. Overall the results are good but see negative values for kappa coefficient and one for human silhouette.

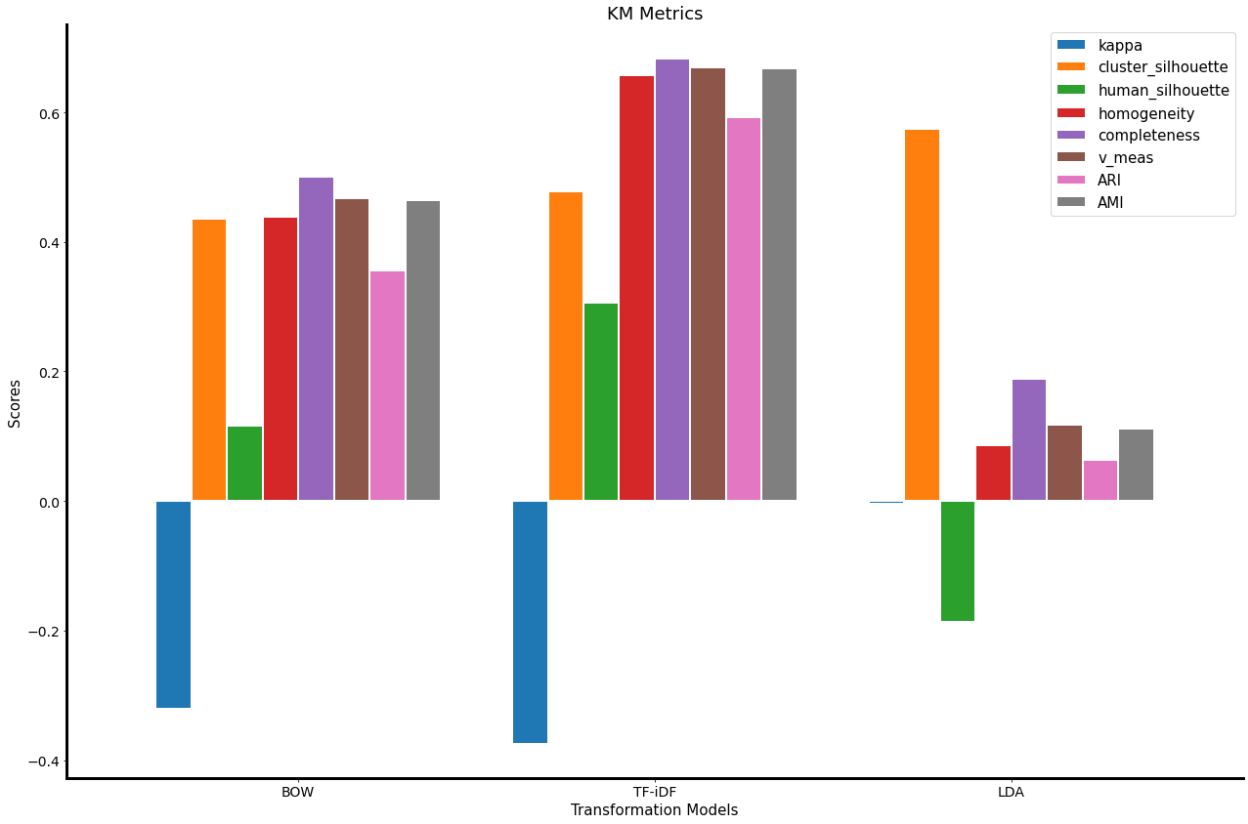


Fig 5.10

Results of Matrix for EM clustering:

The below table shows the error analysis EM clustering Algorithm, We can see the different parameters that can help us to understand the efficiency of the model.
We can see that the silhouette coefficient is highest for the LDA model while it is lowest for the BOW model. Whereas, we can observe poor homogeneity and completeness for the BOW model and good for the Tf-Idf model. Also, we can observe that the kappa coefficients are negative for BOW and TF-iDf models which indicates that clustering model have not efficiently clusters the data points.

	kappa	cluster_silhouette	human_silhouette	homogeneity	completeness	v_meas	ARI	AMI	correlation	pvalue
BOW	-0.313184	0.373399	0.116576	0.392324	0.42683	0.408851	0.300681	0.405746	-0.323884	7.44719e-26
TF-iDF	-0.262431	0.470143	0.305864	0.665675	0.686831	0.676088	0.612186	0.674433	-0.234489	5.8726e-14
LDA	0.294998	0.493751	-0.153249	0.279566	0.356818	0.313503	0.225429	0.309599	0.557176	1.26185e-82
WE	--	0.597140	--	--	--	--	--	--	--	--

Table 5.11

Visualization for Error Analysis Matrix for EM:

The below graph displays the results of all the metrics we used to analyse the EM clustering. Overall the results are acceptable but see negative values for kappa coefficient, human silhouette and correlation.

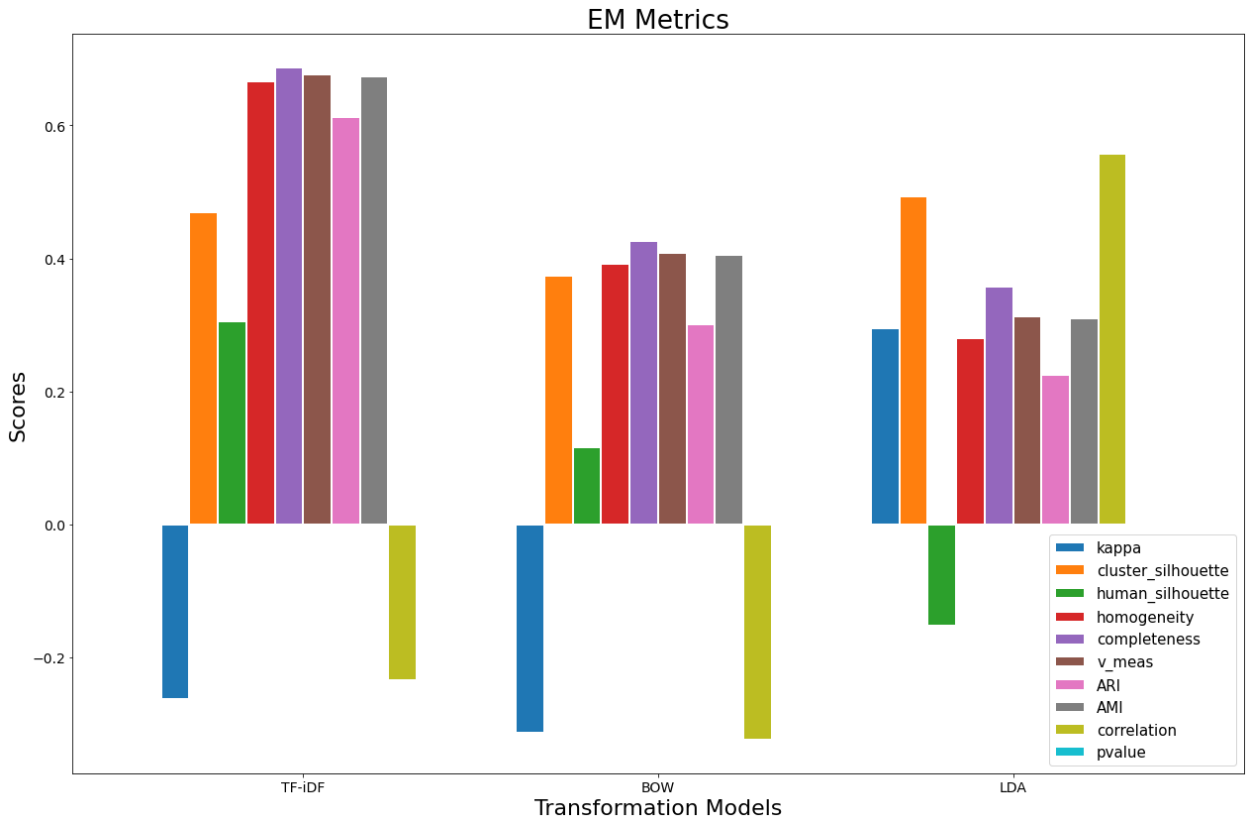


Fig 5.12

6. Analysis using the top 10 most frequent words

While implementing the LDA topic modelling(using gensim) we could analyse the results using the top ten most frequently used words by passing num_words = 10. This provides information regarding the top words which came out of the total number of topics which is 5 in our case. Later we can use this information to visualise the results to an interactive graph and analyse our findings.

Code for implementation:

Below we see the code for the top ten words if the number of topics is 5.

```
# Select the model and print the topics
optimal_model = model_list[3]
model_topics = optimal_model.show_topics(formatted=False)
optimal_model.print_topics(num_words=10)
```

fig6.1

```
[(0,
  '0.069*"could" + 0.047*"attend" + 0.047*"restraint" + 0.047*"sentiment" + 0.047*"unreserve" + 0.047*"aim" + 0.045*"illaudable" + 0.019*"marianne" + 0.017*"said" + 0.011*"might"'),
 (1,
  '0.024*"one" + 0.021*"must" + 0.021*"much" + 0.019*"time" + 0.016*"think" + 0.016*"know" + 0.014*"thing" + 0.014*"edward" + 0.013*"see" + 0.012*"miss"'),
 (2,
  '0.066*"appeared" + 0.034*"would" + 0.012*"good" + 0.010*"yet" + 0.010*"thou" + 0.009*"lucy" + 0.009*"thy" + 0.009*"sure" + 0.008*"house" + 0.008*"however"'),
 (3,
  '0.057*"real" + 0.054*"abhorred" + 0.054*"disgrace" + 0.052*"concealment" + 0.027*"mr" + 0.016*"though" + 0.016*"sister" + 0.013*"day" + 0.012*"mother" + 0.012*"shall"'),
 (4,
  '0.037*"elinor" + 0.025*"every" + 0.017*"well" + 0.012*"dashwood" + 0.012*"willoughby" + 0.012*"first" + 0.011*"john" + 0.011*"colonel" + 0.011*"ever" + 0.010*"jennings"')]
```

Visualisation:

In the below visualization, we notice that we see the top ten most frequent words present for all five topics in the text.

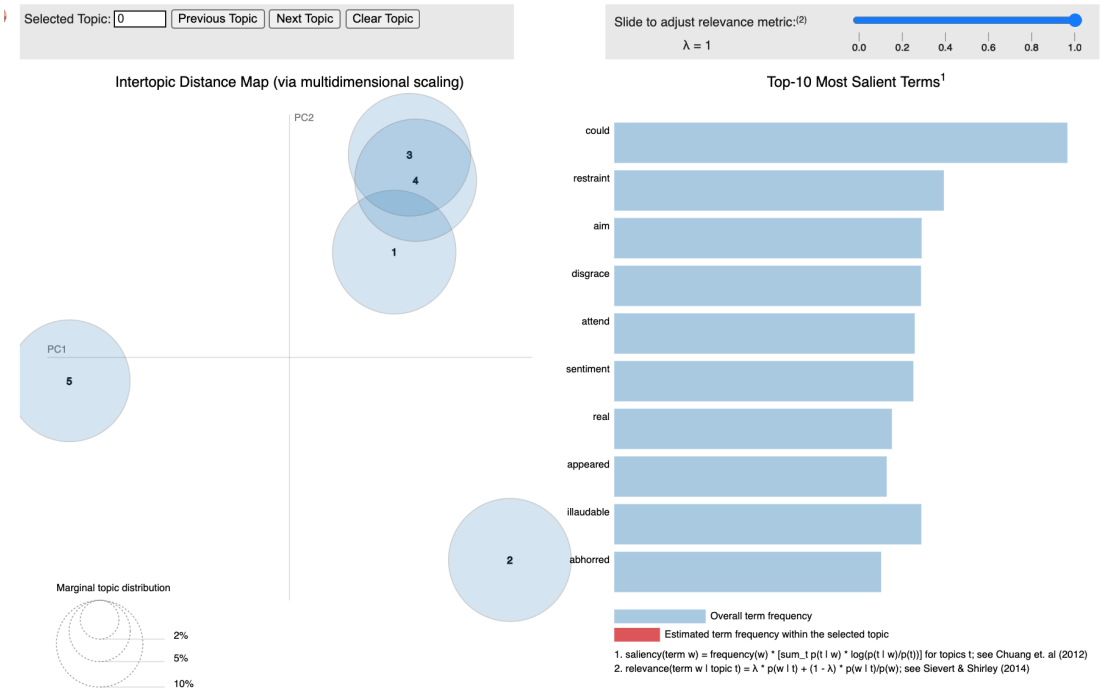


fig6.2

We now see all five topics with the top words of each topic. We notice that the words are in the descending order of frequency and the highly used word is seen at the top of the list. Therefore, we can infer that the top ten words seen in the visualise denotes the top ten most frequently used words in our text for the selected topic. We also noticed that we see the percentage of the top ten words for the selected topic specified in the visualisation. Example, when selecting topic 4, the most relevant top words take 19.8% of total tokens present.

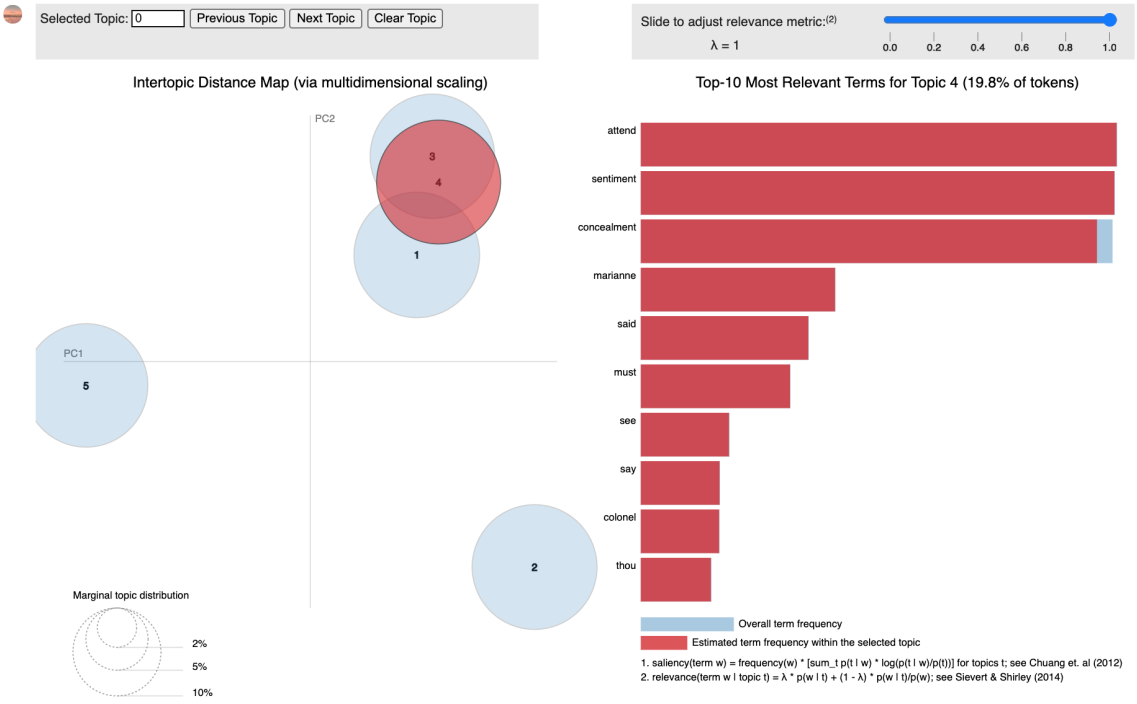

Selected Topic: 0
Previous Topic
Next Topic
Clear Topic
Slide to adjust relevance metric:(2)
λ = 1
0.0
0.2
0.4
0.6
0.8
1.0
Intertopic Distance Map (via multidimensional scaling)
PC2
PC1
1
2
3
4
5
Marginal topic distribution
2%
5%
10%
Top-10 Most Relevant Terms for Topic 4 (19.8% of tokens)
attend
sentiment
concealment
marianne
said
must
see
say
colonel
thou
Overall term frequency
Estimated term frequency within the selected topic
1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (2012)
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

Fig 6.3

7. Insight

- ➔ LDA topic modelling using gensim library uses a lot of RAM, so we tried implementing it on a hierarchical model, the compiler crashes due to lack of RAM.
- ➔ Skip-gram performs better in word embedding than other types.
- ➔ From our analysis, it is safe to say that the EM model using word embedding worked best for clustering and can be considered as our champion model.
- ➔ We get slightly different clustering results every time we run the clustering algorithms. One of the major disadvantages of this is identifying the accurate number of clusters using dendrograms in hierarchical clustering.
- ➔ We also get different results when we run the topic modelling algorithms.
- ➔ The clustering algorithms perform very differently if the number of clusters is changed to another value. Therefore, it is important to look closely while choosing the accurate number.
- ➔ There is no exact way of knowing where to cut the dendrogram to calculate the number of clusters. The accuracy in calculating this comes with more practice.
- ➔ When we implemented the LDA model with the number of clusters equal to 10 the topics were overlapping each other but with the number of topics reduced to 5, this problem was drastically improved and provided us with scattered points.
- ➔ Negative kappa means that the model is not usable and is producing the wrong cluster.