Master of Engineering Electrical and Computer Engineering 2021

University of Ottawa

uOttawa

Clustering Assignment

Authors: Seifeldin Abdelghany, Sara Abdelghafar, Omar Sorour and Zeyad Elsayed

Supervisor: Dr. Arya Rahgozar

Group Number: 8

Submission date: June 20th, 2021

Abstract

Recently, the rise of big data and natural language processing algorithm has gained a huge amount interest from the competing technology companies. Text clustering is one of the problems solved by machine learning algorithm. Text clustering refers to the process of unsupervised learning of specified text based on fixed rules. This report describes the various techniques of text clustering, including text representation, feature selection and clustering algorithms, and draws the basic ideas, advantages and disadvantages of several current mainstream clustering techniques.

Contents

List of Figures

Chapter 1

Introduction

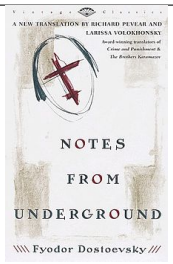
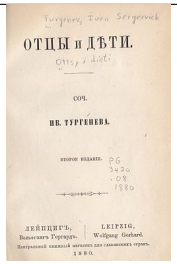
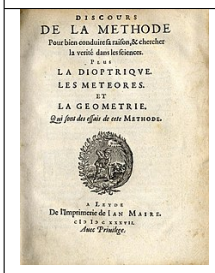
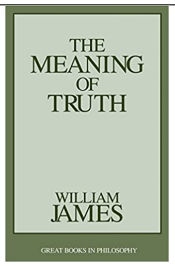
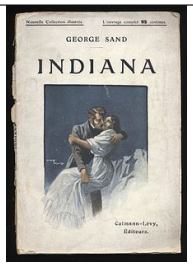
Text clustering can be simply explained as the process of passing the unknown text through some rules to obtain in which cluster the text belongs to. Assuming that there is an objective function, the training of the classification method is carried out through a large amount of corpora, and then the clustering is trained to obtain the model, and then according to the clustering that has been obtained, the various feature item sets are sorted into the initially defined group number. Text clustering is a Machine Learning technique that involves the grouping of data points [1].

1.1 Problem Statement

There are a lot of different philosophical schools. Each of them has it's own thought of existence, life, the values that humans should have, and many other philosophical thoughts. The idea and the thoughts of each school are reflected in the writings and the novels of the authors who belonged to one specific school. Since there are many philosophical schools and thousands of books that reflect the ideas of each

school, labeling any documents and group them to one specific school becomes a challenge. In this research, we will try to use clustering techniques to automatically label documents quoted from five different books that are semantically similar and each book represents one of the philosophical schools, as in table 1.1. We will also try different clustering algorithms along with different feature extracting techniques and feature reduction techniques and determine the best combination that achieves the goal based specific evaluation criteria.

Table 1.1: The books that we used and their philosophical schools.

Image	Book	Author	philosophical school
	Notes from the Underground	Feodor Dostoevsky	Existentialism
	Fathers and Sons	Ivan Turgenev	Nihilism
	A Discourse on Method	René Descartes	Rationalism
	The Meaning of Truth	William James	Pragmatism
	Indiana	George Sand	Idealism

Chapter 2

Code Implementation

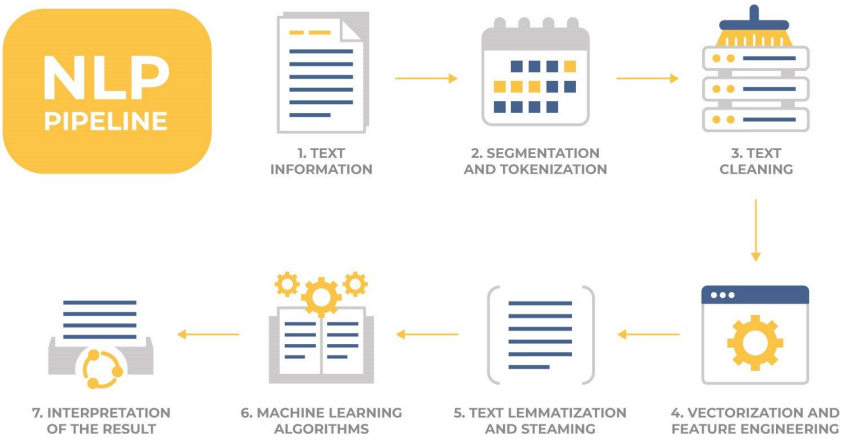


Figure 2.1: NLP Pipeline

To cluster the text, it should pass through modeling process as in Fig 2.1.

2.1 Setting up the Environment

2.1.1 Importing Libraries

```
import matplotlib.pyplot as plt
import nltk
from nltk.probability import FreqDist
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
from collections import Counter
import random
from random import randrange
from random import sample
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sn
nltk.download('gutenberg')
nltk.download('punkt')
nltk.download('stopwords')
stop_words=set(stopwords.words("english"))
```

Figure 2.2: Environment

Starting by setting up our environment, so we imported some libraries to help us in coding, as in Fig 2.2.

2.1.2 Importing Data

```
import glob
import os
books = []
path=r'/content/'
def read_all_text_in_folder(path):
    os.chdir(path)
    myFiles = glob.glob('*.txt')

    for i in myFiles:
        finalpath=path+i
        with open (finalpath) as fo:
            data = fo.read()

        books.append(data)


read_all_text_in_folder(path)
books
```

```
['Title: A Discourse on Method\n\nAuthor: René Descartes\n\n\n\n
 'Title: The Meaning of Truth\n\nAuthor: William James\n\n\nTHE MEANI
 'Title: Indiana\n\nAuthor: George Sand\n\n\nThe Masterpieces of Geor
 'Title: Fathers and Sons\n\nAuthor: Ivan Sergeevich Turgenev\n\n\nFA
 'Title: Notes from the Underground\n\nAuthor: Feodor Dostoevsky\n\n\
```

Figure 2.3: Importing Dataset from Gutenberg

We imported five books from Gutenberg. We chose five books that are semantically similar and each book represents one of the philosophical schools of five different authors, in Fig 2.3. The five books we chose were A Discourse on method, The meaning of truth, Indiana, Fathers and sons and Notes from the underground.

2.2 Cleaning, Partitioning and Labeling Data

Preprocessing and Data Cleansing

```
stemmer = PorterStemmer()
def lemmatize_stemming(text): #lemmatize function
    return stemmer.stem(WordNetLemmatizer().lemmatize(text, pos='v'))

#This function parses the text that we imported along with the number of partitions (Rows) and number of words per each partition (W)
#We can change these numbers each time to evaluate the performance of the classifiers

def get_df(texts,Rows,W): #function that takes the list of books and returns a df with 200 cleaned samples for each book
  filtered_sentences = []
  labels = []
  for book in books: # get the title of each book to be used as label
    x = book.index('\n')
    labels.append(book[7:x]) #parse the title of the book
  label = []
  for i in range(len(texts)):
    text = texts[i] #get the book
    text = text.lower()
    tokenized_word=nltk.word_tokenize(text) #tokenize the words
    cleaned_words = [word for word in tokenized_word if word.isalnum()] #clean the words from symbols and keep alphanumeric characters instead of using regex
    filtered_words=[]
    for w in cleaned_words:# Clean the words yet again from stop words.
        if w not in stop_words:
            filtered_words.append(w)
    for s in range(len(filtered_words)):
      filtered_words[s] = lemmatize_stemming(filtered_words[s])
    random_numbers = []
    for c in range(Rows): # generate this loop 200 times for each book to take 200 samples from ech book
      while True: # To stay in loop and change random number if it was used before
        x = randrange(0,len(filtered_words)-W) #generate random number from 0 to length of string - W (to be able to take last W amount of words)
        if (x not in random_numbers): # make sure the number was not used before
          random_numbers.append(x) #add random number to list
          filtered_sentences.append(' '.join(filtered_words[x:x+W])) #add W amount of words from random position to random position + W.
          label.append(labels[i]) #append the label of the book
          break

  print(len(filtered_sentences))
  print(len(filtered_words))
  df = pd.DataFrame({'label': label, 'sample': filtered_sentences})

  return df, filtered_sentences
df, s = get_df(books,200,150) #Changing the number of partitions and the number of words in every partition will change the whole performance of the experiment.
print(len(s[0]))
print(len(nltk.word_tokenize((s[0]))))
```

Figure 2.4: Preparing Data

We cleaned the words from symbols and removed any garbage characters by keeping alphanumeric characters only and then we applied stemming and lemmatization on the data, lastly we removed any stop words from text.

Then we performed partitioning, by taking random 200 partition from each book, each partition consists of 150 words. We did this by tokenizing text to transform

sentences into words using nltk word tokenize then we appended these words to form a sentence.

Then we performed labeling by appending Book's name to the list of labels simultaneously with the appending of sentence as in Fig 2.4.

This function returns a dataframe of one thousand records(200 for each of the 5 books) and two columns, the first is for sentences and the second is the label so that later we compare it with the clusters, as in Fig 2.5.

	label	sample
0	A Discourse on Method	differ qualiti heaven star think say enough re...
1	A Discourse on Method	natur task set ascertain true method arriv kno...
2	A Discourse on Method	fall age foundat insecur way exampl persuad wo...
3	A Discourse on Method	line could find object simpl capabl distinctli...
4	A Discourse on Method	reckon fals probabl scienc inasmuch borrow pri...
...	...	...
995	Notes from the Underground	long time ago know parent know noth noth whate...
996	Notes from the Underground	stuff soft butter long fine sentiment may well...
997	Notes from the Underground	say vulgar contempt drag public tear transport...
998	Notes from the Underground	intellig highli develop refin feel cours fli c...
999	Notes from the Underground	happi daughter shall say look monster hollow c...

1000 rows × 2 columns

Figure 2.5: Dataframe

2.3 Text Transformation

We used five transformation techniques,

- BOW
- TF-IDF
- N-gram
- LDA
- Doc2Vec

2.3.1 BOW

```
count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(sample)
X_train_counts.shape
# #print(X_train_counts)
```

```
(1000, 7819)
```

Figure 2.6: BOW transformation

A bag of words is a representation of text that describes the occurrence of words within a document. We used sklearn count vectorizer which converts a collection of text documents to a matrix of token counts as in Fig 2.6.

2.3.2 TF-IDF

```
from sklearn.feature_extraction.text import TfidfTransformer
#vectoriztion before claculating the TF-IDF
vectorizer = TfidfVectorizer()
X_train_counts= vectorizer.fit_transform(sample)
#claculating the TF-IDF values
tfidf_transformer = TfidfTransformer(use_idf=False).fit(X_train_counts)
X_train_tfidf = tfidf_transformer.transform(X_train_counts)
X_train_tfidf.shape
```

```
(1000, 7819)
```

Figure 2.7: TF-IDF transformation

Bag of word doesn't capture the importance of the word it gives you the frequency of the word. TF-IDF resolves this matter through computation of two values. Tf is count of occurrences of the word in a document. IDF of the word across a set of documents. It tells us how common or rare a word is in the entire document set. The closer it is to 0, the more common is the word. This metric can be calculated by taking the total number of documents, dividing it by the number of documents that contain a word, and calculating the logarithm. We then multiply these two values TF and IDF. We used sklearn tfidf-transformer which transforms a count matrix to a normalized tf or tf-idf representation as in Fig 2.7.

2.3.3 N-gram

```
count_vect_ngram = CountVectorizer(ngram_range=(3,3))
X_train_ngram = count_vect_ngram.fit_transform(sample)
X_train_ngram.shape
```

```
(1000, 84187)
```

Figure 2.8: N-gram transformation

Given a sequence of N-1 words, an N-gram model predicts the most probable word that might follow this sequence. It's a probabilistic model that's trained on a corpus of text. An N-gram model is built by counting how often word sequences occur in corpus text and then estimating the probabilities. Since a simple N-gram model has limitations, improvements are often made via smoothing, interpolation and backoff. We used sklearn countvectorizer giving it parameter for n-gram range which is min and max number of the word sequence as in Fig 2.8.

2.3.4 LDA

```
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.datasets import make_multilabel_classification
#vectoriztion before claculating the TF-IDF
vectorizer = TfidfVectorizer()
X_train_counts= vectorizer.fit_transform(sample)
#claculating the TF-IDF values
lda = LatentDirichletAllocation(n_components=30,random_state=123)
lda.fit(X_train_counts)
X_train_LDA = lda.transform(X_train_counts)
# getting the keywords frequency
oo=dict(zip(vectorizer.get_feature_names(), X_train_tfidf.toarray()[0]))
X_train_LDA.shape
```

```
(1000, 30)
```

Figure 2.9: LDA transformation

the Latent Dirichlet Allocation (LDA) is a generative statistical model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar. We used sklearn LatentDirichletAllocation, as in Fig 2.9.

2.3.5 Doc2Vec

```
df1=df
df1.index = range(1000)
df1['sample'].apply(lambda x: len(x.split(' '))).sum()
```

```
150000
```

```
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
from sklearn.utils import shuffle
import time
train, test = train_test_split(df1, test_size=0.00001, random_state=42)
import nltk
from nltk.corpus import stopwords
def tokenize_text(text):
    tokens = []
    for sent in nltk.sent_tokenize(text):
        for word in nltk.word_tokenize(sent):
            if len(word) < 2:
                continue
            tokens.append(word.lower())
    return tokens
train_tagged = train.apply(
    lambda r: TaggedDocument(words=tokenize_text(r['sample']), tags=[r.label]), axis=1)
test_tagged = test.apply(
    lambda r: TaggedDocument(words=tokenize_text(r['sample']), tags=[r.label]), axis=1)
```

```
import multiprocessing
cores = multiprocessing.cpu_count()
```

```
from tqdm import tqdm
model_dbow = Doc2Vec(dm=0, vector_size=300, negative=5, hs=0, min_count=2, sample = 0, workers=cores)
model_dbow.build_vocab([x for x in tqdm(train_tagged.values)])
```

```
100%|██████████| 999/999 [00:00<00:00, 515388.65it/s]
```

Figure 2.10: N-gram

Paragraph Vector (Doc2Vec) is supposed to be an extension to Word2Vec such that Word2Vec learns to project words into a latent d-dimensional space whereas Doc2Vec aims at learning how to project a document into a latent d-dimensional space. We first train the model and then victories the data. We used Doc2Vec, as in Fig 2.10.

2.4 Clustering

We used three clustering techniques to grouping the sentence to it's corresponding author using different transformation techniques,

- k-means
- Hierarchical clustering algorithms
- Expectation Maximization

2.4.1 K-means

We wrote a k-mean function to call it with each of the transformation techniques as in Fig 2.11

```
from sklearn.metrics import silhouette_score
def kmeans_reduction(reduction, x, y, flag = None): #function that uses kmeans to cluster the data based on three types of reduction techniques.
  if reduction == "LDA":
    LDA = LinearDiscriminantAnalysis(n_components= 2)
    if flag == None:
      X = LDA.fit_transform(x.toarray(),y)
    else: X = LDA.fit_transform(x,y)
    plt.scatter(X[:,0], X[:,1])
    plt.title("LDA scatter")
    plt.show()
  if reduction == "tsne":
    tsne = TSNE(n_components=2, random_state=0)
    X = tsne.fit_transform(x)
    plt.scatter(X[:,0], X[:,1])
    plt.title("t-sne scatter")
    plt.show()
  if reduction == "normal":
    X = x
  if reduction == "svd":
    svd = TruncatedSVD(n_components=2, n_iter=7, random_state=42)
    X = svd.fit_transform(x)
    plt.scatter(X[:,0], X[:,1])
    plt.title("SVD scatter")
    plt.show()
  kmeans = KMeans(n_clusters=5, init='k-means++', max_iter=1000, n_init=10, random_state=0)
  pred_y = kmeans.fit_predict(X)
  if reduction != "normal":
    plt.scatter(X[:,0], X[:,1])
    plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=20, c='red')
    plt.title('Clusters')
    plt.show()
  labels = kmeans.labels_
  print("\n Silhouette Score: ", metrics.silhouette_score(X, labels, metric='euclidean') )
  print("\n Contingency Matrix: \n", contingency_matrix(pred_y, y))
```

Figure 2.11: K-mean function.

To make sure that the K-mean model is working correctly, we performed the elbow method and the elbow was at 5 clusters and we have 5 labels so it is right, as in Fig 2.12.

```python
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11): #plot elbow method to find best k -> number of cluster
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=6, random_state=0)
    kmeans.fit(X_train_counts)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss, marker='o')
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.grid(True)
plt.show()
```

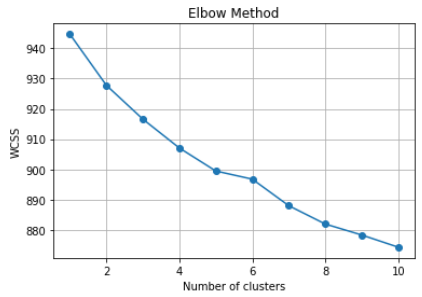


Figure 2.12: Elbow Method.

K-means with BOW.

```python
kmeans_reduction("normal", X_train_counts, y_train)
```

```
Silhouette Score:  0.024170915779075174

Contingency Matrix:
[[ 28 172   0   0   0]
[  0   1 197   0   2]
[  0   2   0 197   1]
[  0  15   8   0 377]]

Kappa score:  0.921875
```

Figure 2.13: K-mean with BOW

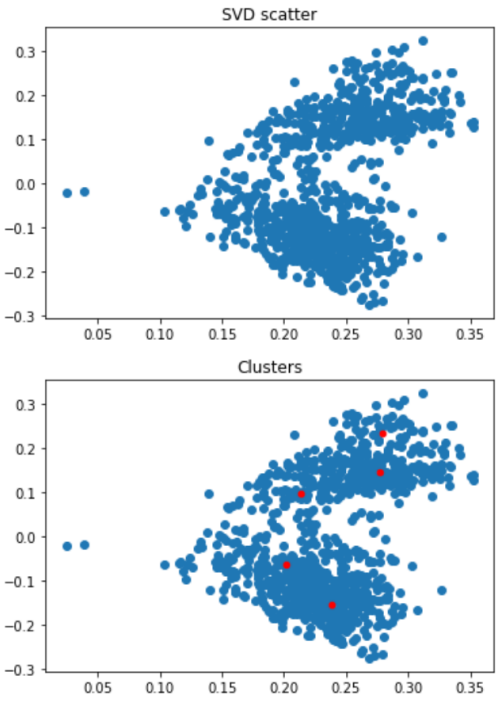
The BOW transformation technique was used on the training data then was fed to the k-mean model. The model was then fitted as seen in Fig 2.13. The high kappa score indicated that we can depend on this model with a percentage of 92 %. However, the low silhouette score indicated that the points in each cluster are far away from each other and the clusters are somewhat close to each other.

Therefore, we decided to perform feature reduction to enhance and improve our model.

- feature reduction allow us to plot 2D graph of the documents.
- enhance the clustering separation.
- reducing the run time.

Feature Reduction using SVD

```
kmeans_reduction("svd", X_train_counts, y_train)
```

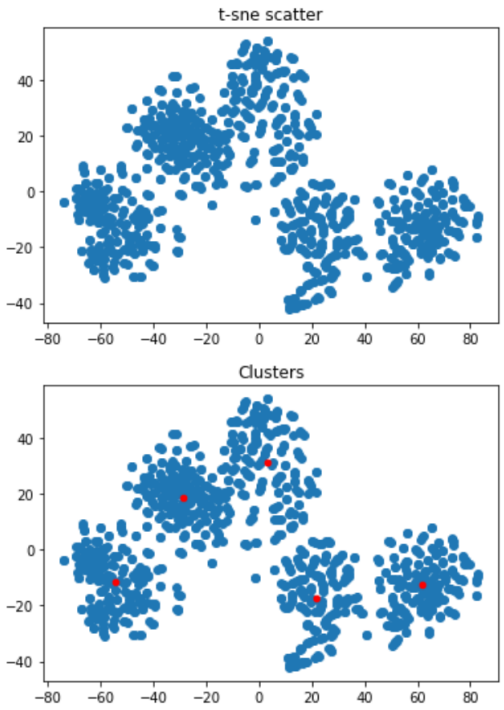



```
Silhouette Score:  0.4066026738174163

Contingency Matrix:
[[133  13   0  26  28]
[  0 103  84  13   0]
[  0 155 244   1   0]
[ 61   1   0  83  55]]

Kappa score:  0.4181091877496671
```

Figure 2.14: K-mean with BOW with SVD

Feature Reduction using T-SNE

```
kmeans_reduction("tsne", X_train_counts, y_train)
```




```
Silhouette Score:  0.52219236

Contingency Matrix:
[[200   0   0   0   0]
[  0 200   0   0   0]
[  0   2 187   0  11]
[  0   2   1 197   0]
[  0   2   1   0 197]]

Kappa score:  0.97625
```

Figure 2.15: K-mean with BOW with T-SNE

The best silhouette score and kappa in kmean and BOW was by T-SNE, as in Fig 2.14,2.15.

K-means with TF-IDF.

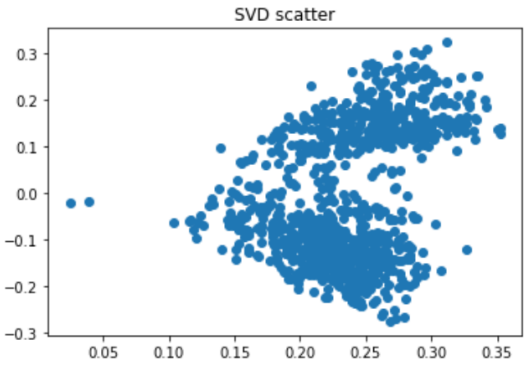
```
kmeans_reduction("normal", X_train_tfidf, y_train)

Silhouette Score:  0.024170915779075177

Contingency Matrix:
[[ 28 172   0   0   0]
[  0   1 197   0   2]
[  0   2   0 197   1]
[  0  15   8   0 377]]

Kappa score:  0.921875
```

Figure 2.16: K-mean with TF-IDF

The TF-IDF transformation technique was used on the training data then was fed to the kmeans model. The model was then fitted as seen in Fig 2.16. The high kappa score indicated that we can depend on this model with a percentage of 98 %. However, the low silhouette score indicated that the points in each cluster are far away from each other and the clusters are somewhat close to each other. Therefore, we decided to perform feature reduction to enhance and improve our model.

Feature Reduction using SVD

```
kmeans_reduction("svd", X_train_tfidf, y_train)
```

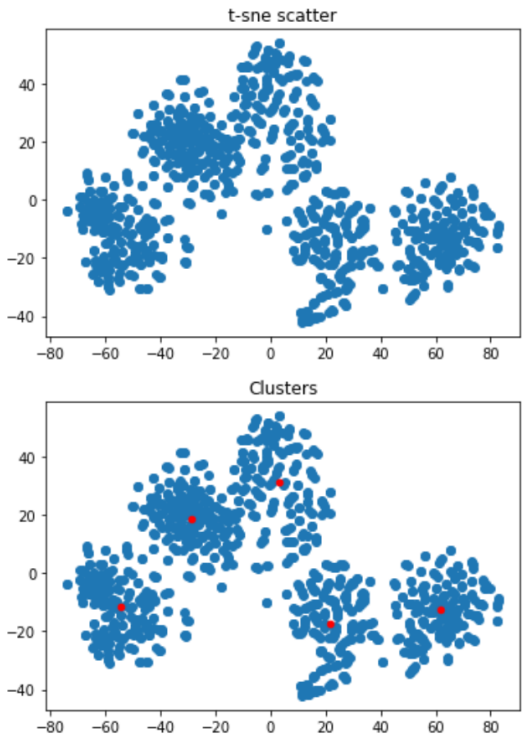



```
Silhouette Score:  0.40660267381741655

Contingency Matrix:
[[133  13   0  26  28]
[  0 103  84  13   0]
[  0 155 244   1   0]
[ 61   1   0  83  55]]

Kappa score:  0.4181091877496671
```

Figure 2.17: K-mean with TF-IDF with SVD

Feature Reduction using T-SNE

```
kmeans_reduction("tsne", X_train_tfidf, y_train)
```




```
Silhouette Score:  0.52219236

Contingency Matrix:
[[200   0   0   0   0]
[  0 200   0   0   0]
[  0   2 187   0  11]
[  0   2   1 197   0]
[  0   2   1   0 197]]

Kappa score:  0.97625
```

Figure 2.18: K-mean with TF-IDF with T-SNE

The best silhouette score and kappa in kmean and TF-IDF was by T-SNE, as in Fig 2.17,2.18.

K-means with n-gram.

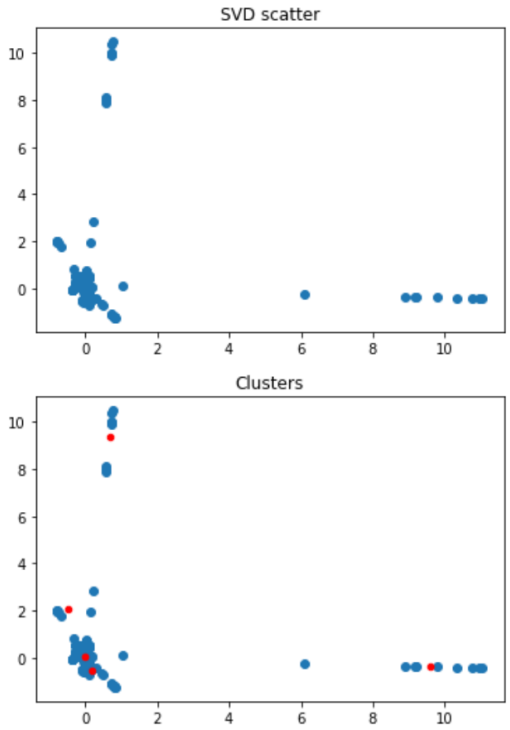
```
kmeans_reduction("normal", X_train_ngram, y_train)

Silhouette Score:  0.008594092503700251

Contingency Matrix:
[[ 10   4 971   8   7]]

Kappa score:  0.0
```

Figure 2.19: K-mean with n-gram

The n-gram transformation technique was used on the training data then was fed to the kmeans model. The model was then fitted as seen in Fig 2.19. The low kappa score indicated that we can not depend on this model with a percentage of 0 %. However, the low silhouette score indicated that the points in each cluster are far away from each other and the clusters are somewhat close to each other. Therefore, we decided to perform feature reduction to enhance and improve our model.

Feature Reduction using SVD

```
kmeans_reduction("svd", X_train_ngram, y_train)
```

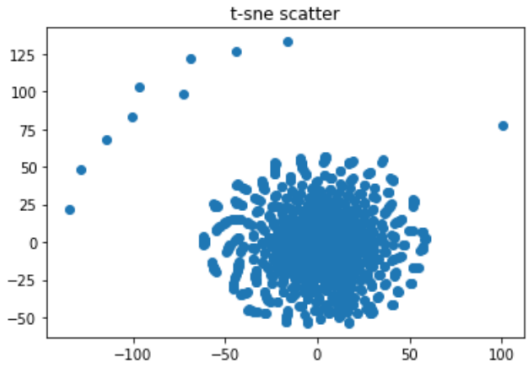



```
Silhouette Score:  0.8227772755351176

Contingency Matrix:
[[926   9   8   7  50]]

Kappa score:  0.0
```

Figure 2.20: K-mean with n-gram with SVD

Feature Reduction using T-SNE

```
kmeans_reduction("tsne", X_train_ngram, y_train)
```




```
Silhouette Score:  0.3393772

Contingency Matrix:
[[ 95   3  15   5  82]
[ 53 188  59  70  30]
[  3  67  97   0  33]
[ 18  45   8 123   6]]

Kappa score:  0.3542099792099792
```

Figure 2.21: K-mean with n-gram with T-SNE

The best silhouette score and kappa in kmean and n-gram was by T-SNE, as in Fig 2.20,2.21.

K-means with LDA.

```
kmeans_reduction("normal", X_train_LDA, y_train)
```

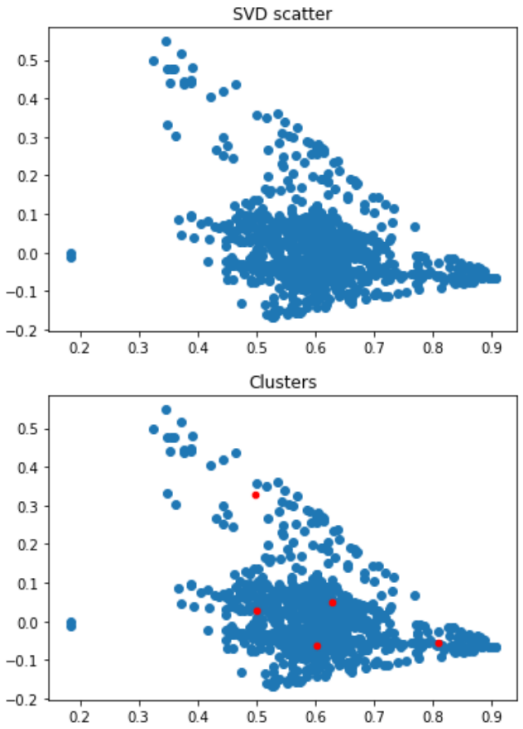
```
Silhouette Score:  0.15455314746366003

Contingency Matrix:
[[114 182  75 558  71]]

Kappa score:  0.0
```

Figure 2.22: K-mean with LDA

The LDA transformation technique was used on the training data then was fed to the kmeans model. The model was then fitted as seen in Fig 2.22. The low kappa score indicated that we can not depend on this model with a percentage of 0 %. However, the low silhouette score indicated that the points in each cluster are far away from each other and the clusters are somewhat close to each other. Therefore, we decided to perform feature reduction to enhance and improve our model.

Feature Reduction using SVD

```
kmeans_reduction("svd", X_train_LDA, y_train)
```

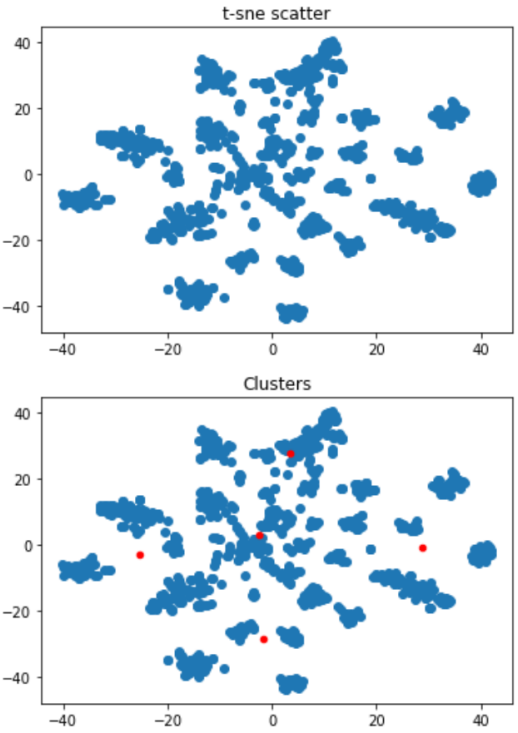



```
Silhouette Score:  0.37768751170118947

Contingency Matrix:
[[284  36  69 102 109]
[ 83  17  40 109 151]]

Kappa score:  0.16395383249482087
```

Figure 2.23: K-mean with LDA with SVD

Feature Reduction using T-SNE

```
kmeans_reduction("tsne", X_train_LDA, y_train)
```




```
Silhouette Score:  0.40838206

Contingency Matrix:
[[ 63  23  54  33  27]
[ 35  56  37  36  36]
[ 46  21  60  50  23]
[ 63  91  58  74 114]]

Kappa score:  0.11470072627097405
```

Figure 2.24: K-mean with LDA with T-SNE

The best silhouette score and kappa in kmean and LDA was by SVD, as in Fig 2.23,2.24.

K-means with Doc2vec.

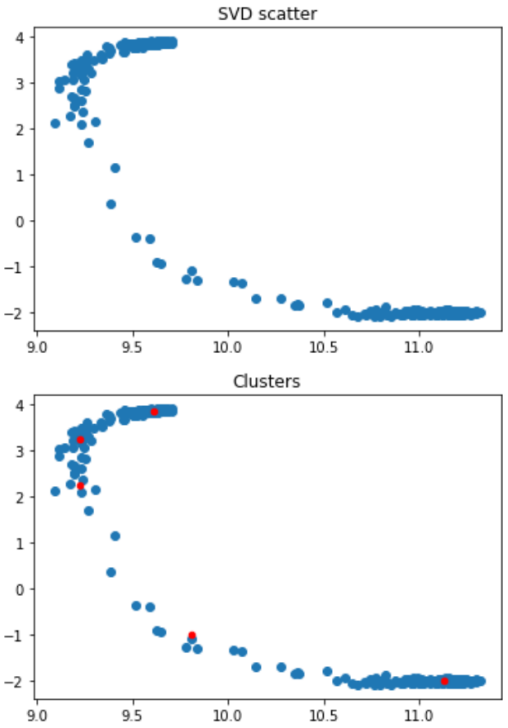
```
kmeans_reduction("normal", X_train_D2V, y_train_D2V)

Silhouette Score:  0.75523055

Contingency Matrix:
[[ 74  60   1   1   9]
[157 361  10  11  16]]

Kappa score:  0.16760152568992592
```

Figure 2.25: K-mean with Doc2vec

The Doc2vec transformation technique was used on the training data then was fed to the kmeans model. The model was then fitted as seen in Fig 2.25. The low kappa score indicated that we can not depend on this model with a percentage of 16 %. However, the high silhouette score indicated that the points in each cluster are not far away from each other and the clusters are somewhat far to each other. Therefore, we decided to perform feature reduction to enhance and improve our model.

Feature Reduction using SVD

```
kmeans_reduction("svd", X_train_D2V, y_train_D2V)
```

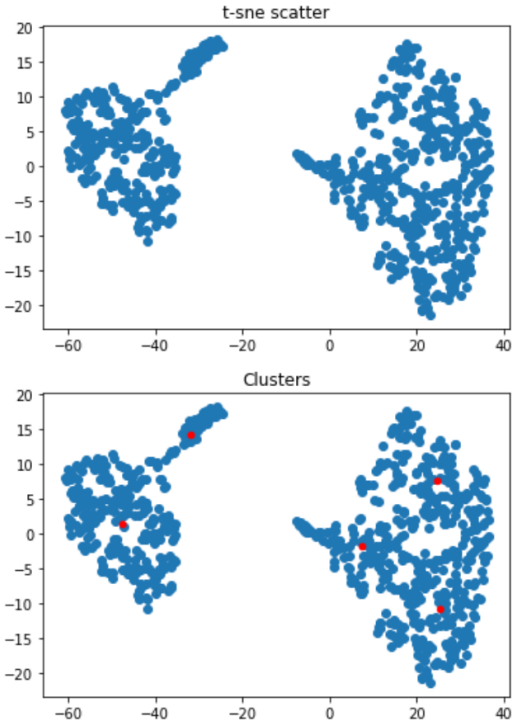



```
Silhouette Score:  0.88177454

Contingency Matrix:
[[ 74  60   2   1   8]
[157 361  10  11  16]]

Kappa score:  0.16760152568992592
```

Figure 2.26: K-mean with Doc2vec with SVD

Feature Reduction using T-SNE

```
kmeans_reduction("tsne", X_train_D2V, y_train_D2V)
```




```
Silhouette Score:  0.46231106

Contingency Matrix:
[[ 80 159  71  36  78]
[ 69  58  50  14  85]]

Kappa score:  0.09576095455009015
```

Figure 2.27: K-mean with Doc2vec with T-SNE

The best silhouette score and kappa in kmean and Doc2vec was by SVD, as in Fig 2.26,2.27.

2.4.2 Hierarchical clustering

We wrote a Hierarchical clustering function to call it with each of the transformation techniques as in Fig 2.28

```
def hc_reduction(reduction, x, y, flag = None): #function that uses hierarchical clustering to cluster the data based on three types of reduction techniques.
  random.seed(555)
  if reduction == "LDA":
    LDA = LinearDiscriminantAnalysis(n_components= 2)
    if flag == None:
      X = LDA.fit_transform(x.toarray(),y)
    else: X = LDA.fit_transform(x,y)
    plt.scatter(X[:,0], X[:,1])
    plt.title("LDA scatter")
    plt.show()
  if reduction == "tsne":
    tsne = TSNE(n_components=2, random_state=0)
    X = tsne.fit_transform(x)
    plt.scatter(X[:,0], X[:,1])
    plt.title("t-sne scatter")
    plt.show()
  if reduction == "normal":
    if flag == None:
      X = x.toarray()
    else: X = x
  if reduction == "svd":
    svd = TruncatedSVD(n_components=2, n_iter=7, random_state=42)
    X = svd.fit_transform(x)
    plt.scatter(X[:,0], X[:,1])
    plt.title("SVD scatter")
    plt.show()
  if reduction != "normal":
    plt.figure(figsize=(10, 7))
    plt.title("Dendrograms")
    dend = shc.dendrogram(shc.linkage(X, method='ward'))
    plt.show()
  hc = AgglomerativeClustering(n_clusters=5, affinity = 'euclidean', linkage = 'ward')
  pred_y = hc.fit_predict(X)
  labels = hc.labels_
  print("\n Silhouette Score: ", silhouette_score(X, labels, metric='euclidean') )

  #This part of the function matches between the cluster label and the original classs by identifing the maximum number of records that hav the same class in each label
  #Then we change the number that represents that class with the same number that represent the label so that we will be able to calculate the kappa and the contengincy matrix
  y= pd.factorize(y)
  correcting_class= pd.DataFrame({"label":pred_y,"class":y[0]})
  lll=[]
  for i in range(5):
    lll.append(correcting_class.loc[correcting_class["class"]==i])
  for i in range(5):
    lll[i]["class"]=max((lll[i]["label"].mode()))
  class_laabel_df=pd.DataFrame()
  for I in range(5):
    class_laabel_df=pd.concat([class_laabel_df, lll[I]], axis=0)
  class_laabel_df
  print("\n Contingency Matrix: \n", contingency_matrix(class_laabel_df["class"], class_laabel_df["label"]))
  print("\n Kappa score: " ,cohen_kappa_score(class_laabel_df["class"], class_laabel_df["label"]))
  print("\n\n")
```

Figure 2.28: Hierarchical clustering function.

Hierarchical clustering with BOW.

```
hc_reduction("normal", X_train_counts, y_train)
```

```
Silhouette Score:  0.02335177273793631

Contingency Matrix:
[[400   0   0   0   0]
[  4 176   4   0  16]
[  3   3 194   0   0]
[ 10   0   0 190   0]]

Kappa score:  0.9444290080577938
```

Figure 2.29: Hierarchical clustering with BOW

The BOW transformation technique was used on the training data then was fed to the Hierarchical clustering model. The model was then fitted as seen in Fig 2.29. The high kappa score indicated that we can depend on this model with a percentage of 94 %. However, the low silhouette score indicated that the points in each cluster are far away from each other and the clusters are somewhat close to each other. Therefore, we decided to perform feature reduction to enhance and improve our model.

Feature Reduction using SVD

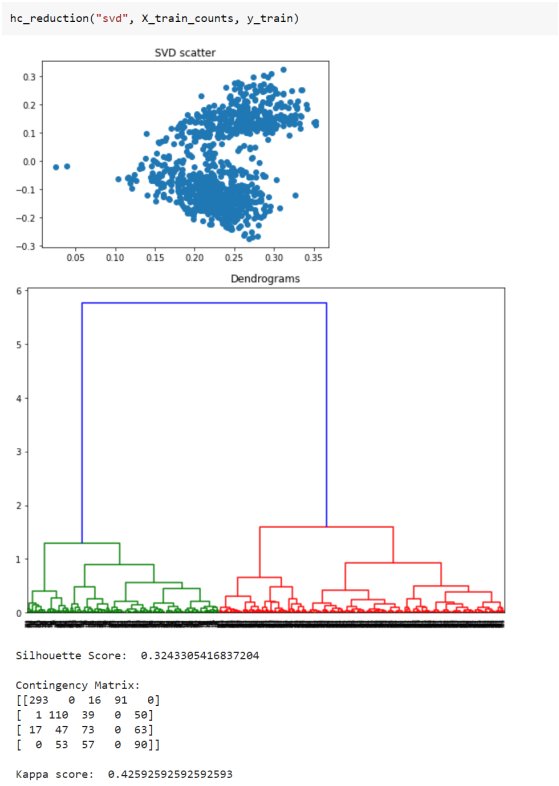


Figure 2.30: Hierarchical clustering with BOW with SVD

Feature Reduction using T-SNE

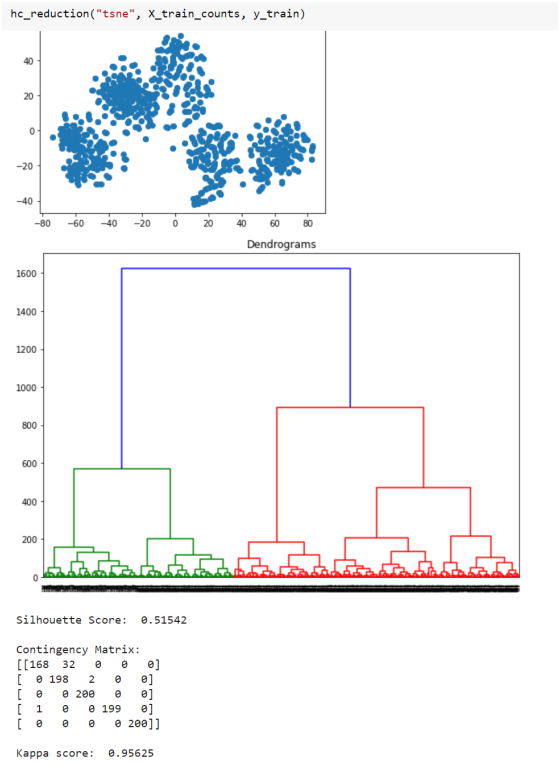


Figure 2.31: Hierarchical clustering with BOW with T-SNE

The best silhouette score and kappa in Hierarchical clustering and BOW was by T-SNE, as in Fig 2.30,2.31.

Hierarchical clustering with TF-IDF.

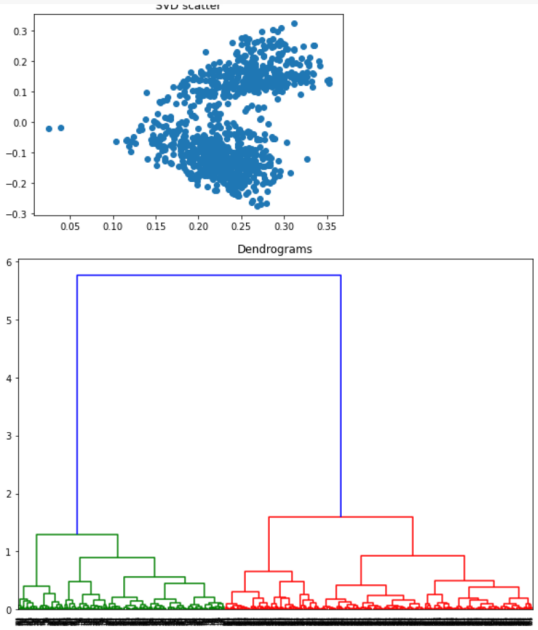
```
hc_reduction("normal", X_train_tfidf, y_train)

Silhouette Score:  0.023351772737936316

Contingency Matrix:
[[400   0   0   0   0]
[  4 176   4   0  16]
[  3   3 194   0   0]
[ 10   0   0 190   0]]

Kappa score:  0.9444290080577938
```

Figure 2.32: Hierarchical clustering with TF-IDF

The TF-IDF transformation technique was used on the training data then was fed to the Hierarchical clustering model. The model was then fitted as seen in Fig 2.32. The high kappa score indicated that we can depend on this model with a percentage of 94 %. However, the low silhouette score indicated that the points in each cluster are far away from each other and the clusters are somewhat close to each other. Therefore, we decided to perform feature reduction to enhance and improve our model.

Feature Reduction using SVD

```
hc_reduction("svd", X_train_tfidf, y_train)
```




```
Silhouette Score:  0.32433054168372055

Contingency Matrix:
[[293   0  16  91   0]
[  1 110  39   0  50]
[ 17  47  73   0  63]
[  0  53  57   0  90]]

Kappa score:  0.42592592592592593
```

Figure 2.33: Hierarchical clustering with TF-IDF with SVD

Feature Reduction using T-SNE

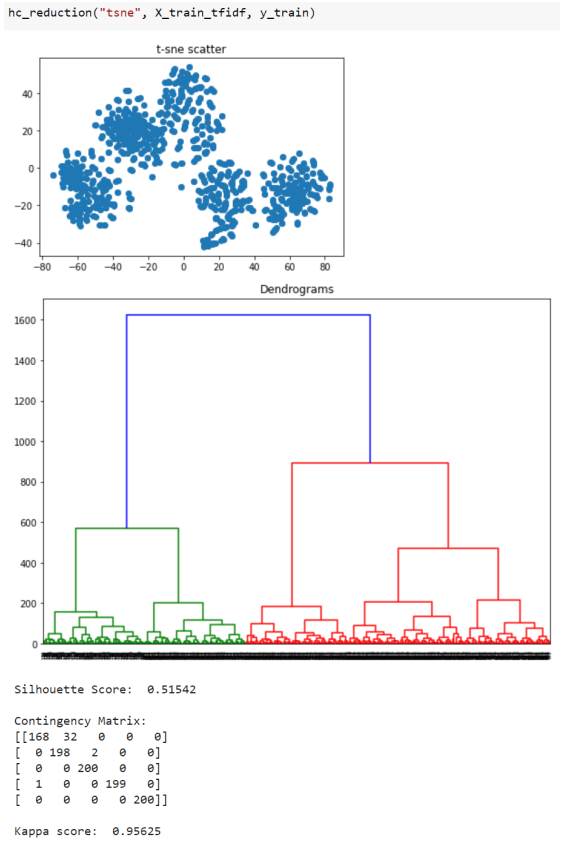


Figure 2.34: Hierarchical clustering with TF-IDF with T-SNE

The best silhouette score and kappa in Hierarchical clustering and TF-IDF was by T-SNE, as in Fig 2.33,2.34.

Hierarchical clustering with n-gram.

```
hc_reduction("normal", X_train_ngram, y_train)

Silhouette Score:  0.010289139473876269

Contingency Matrix:
[[969   8   8   9   6]]

Kappa score:  0.0
```

Figure 2.35: Hierarchical clustering with n-gram

The n-gram transformation technique was used on the training data then was fed to the Hierarchical clustering model. The model was then fitted as seen in Fig 2.35. The low kappa score indicated that we can not depend on this model with a percentage of 0 %. However, the low silhouette score indicated that the points in each cluster are far away from each other and the clusters are somewhat close to each other. Therefore, we decided to perform feature reduction to enhance and improve our model.

Feature Reduction using SVD

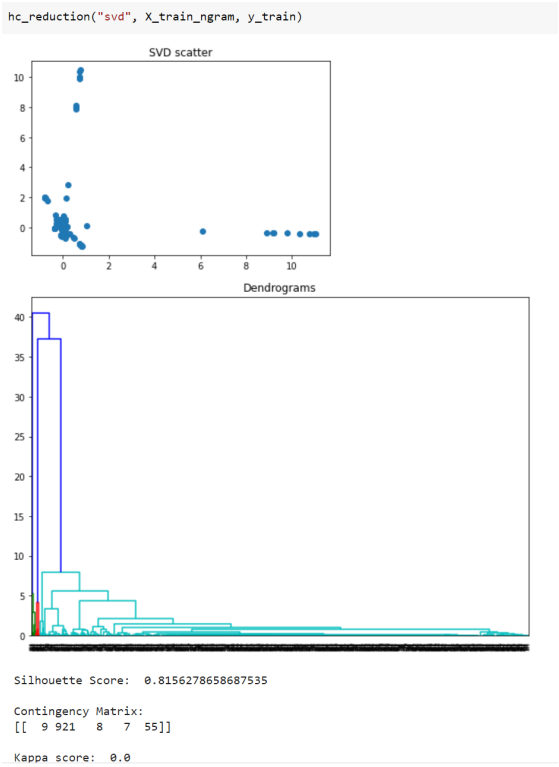


Figure 2.36: Hierarchical clustering with n-gram with SVD

Feature Reduction using T-SNE

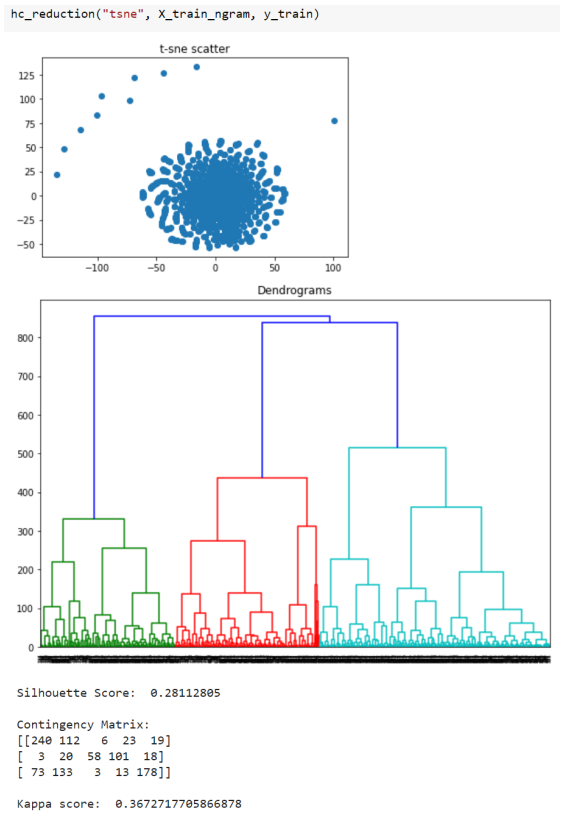


Figure 2.37: Hierarchical clustering with n-gram with T-SNE

The best silhouette score and kappa in Hierarchical clustering and n-gram was by T-SNE, as in Fig 2.36,2.37.

Hierarchical clustering with LDA.

```
hc_reduction("normal", X_train_LDA, y_train, 1)

Silhouette Score:  0.13526064875890847

Contingency Matrix:
[[650  78  78 109  85]]

Kappa score:  0.0
```

Figure 2.38: Hierarchical clustering with LDA

The LDA transformation technique was used on the training data then was fed to the Hierarchical clustering model. The model was then fitted as seen in Fig 2.38. The low kappa score indicated that we can not depend on this model with a percentage of 0 %. However, the low silhouette score indicated that the points in each cluster are far away from each other and the clusters are somewhat close to each other. Therefore, we decided to perform feature reduction to enhance and improve our model.

Feature Reduction using SVD

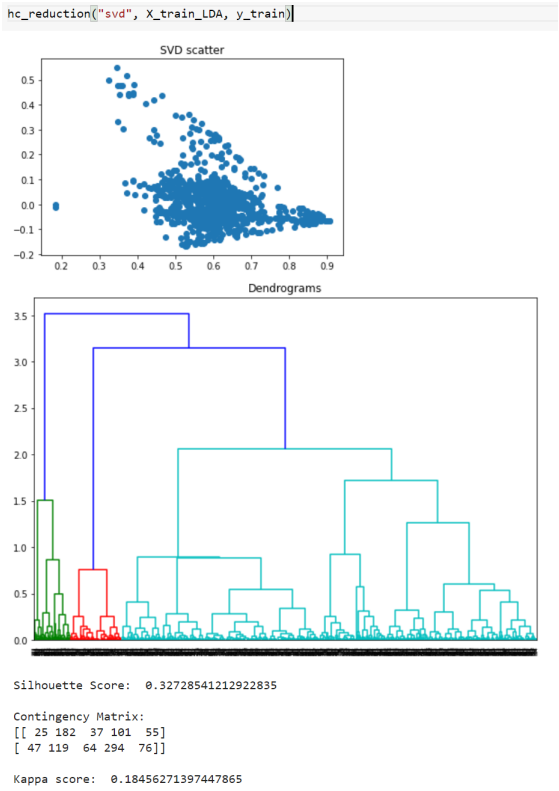


Figure 2.39: Hierarchical clustering with LDA with SVD

Feature Reduction using T-SNE

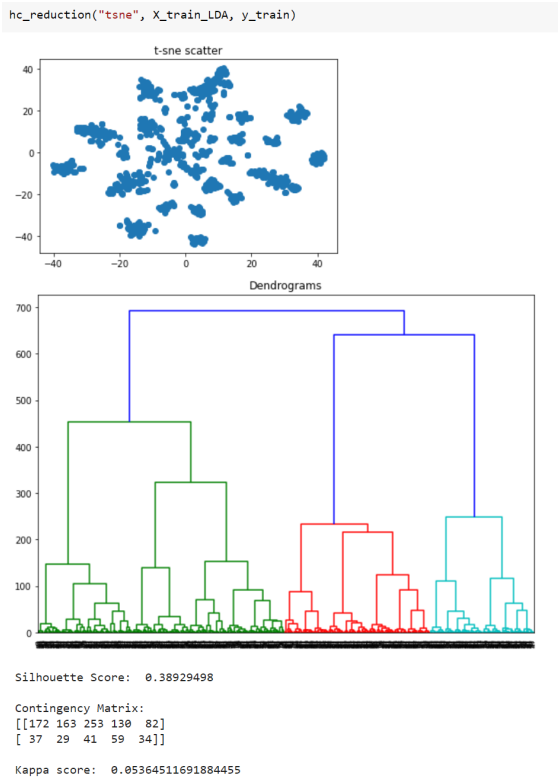


Figure 2.40: Hierarchical clustering with LDA with T-SNE

The best silhouette score and kappa in Hierarchical clustering and LDA was by SVD, as in Fig 2.39,2.40.

Hierarchical clustering with Doc2vec.

```
hc_reduction("normal", X_train_D2V, y_train_D2V, 1)

Silhouette Score:  0.7504924

Contingency Matrix:
[[363  12   9 156  15]
[ 60   3   1  74   7]]

Kappa score:  0.17029091646573669
```

Figure 2.41: Hierarchical clustering with Doc2vec

The Doc2vec transformation technique was used on the training data then was fed to the Hierarchical clustering model. The model was then fitted as seen in Fig 2.41. The low kappa score indicated that we can not depend on this model with a percentage of 17 %. However, the low silhouette score indicated that the points in each cluster are far away from each other and the clusters are somewhat close to each other. Therefore, we decided to perform feature reduction to enhance and improve our model.

Feature Reduction using SVD

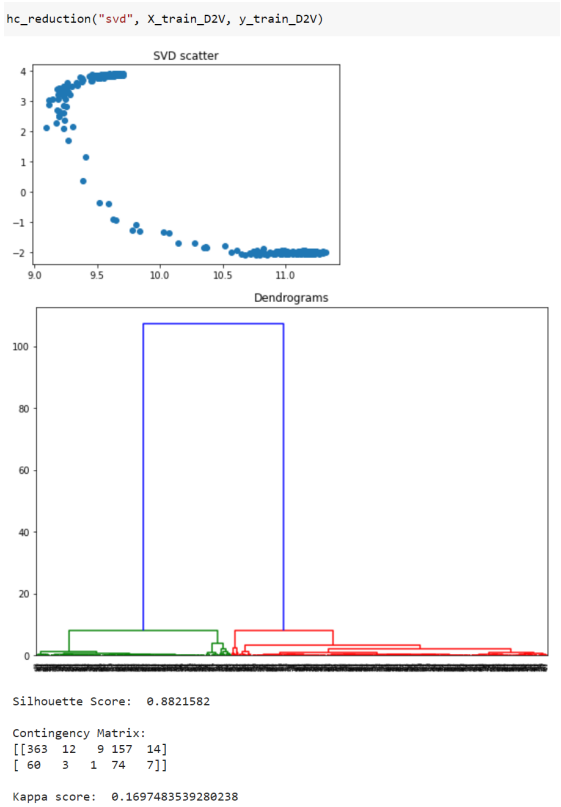


Figure 2.42: Hierarchical clustering with Doc2vec with SVD

Feature Reduction using T-SNE

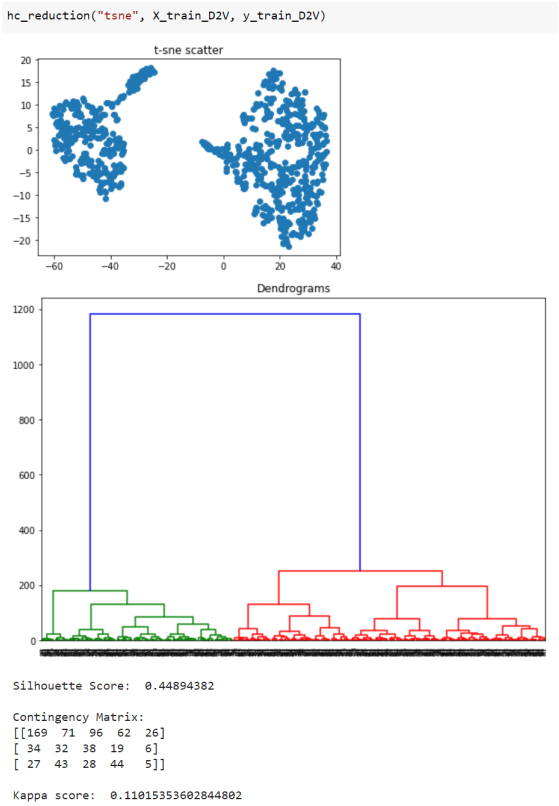


Figure 2.43: Hierarchical clustering with Doc2vec with T-SNE

The best silhouette score and kappa in Hierarchical clustering and Doc2vec was by SVD, as in Fig 2.42,2.43.

2.4.3 Expectation Maximization

We wrote a Expectation Maximization function to call it with each of the transformation techniques as in Fig 2.44

```
def em_reduction(reduction, x, y, flag = None): #function that uses kmeans to cluster the data based on three types of reduction techniques.
  random.seed(555)
  if reduction == "tsne":
    tsne = TSNE(n_components=2, random_state=0)
    X = tsne.fit_transform(x)
    plt.scatter(X[:,0], X[:,1])
    plt.title("t-sne scatter")
    plt.show()
  if reduction == "normal":
    if flag == None:
      X = x.toarray()
    else: X = x
  if reduction == "svd":
    svd = TruncatedSVD(n_components=2, n_iter=7, random_state=42)
    X = svd.fit_transform(x)
    plt.scatter(X[:,0], X[:,1])
    plt.title("SVD scatter")
    plt.show()
  gmm = GaussianMixture(n_components=5)
  gmm.fit(X)
  pred_y = gmm.predict(X)
  Y = np.array([[0.5],[0.5]])
  if reduction != "normal":
    # plt.scatter(X[:,0], X[:,1])
    # plt.scatter(gmm.cluster_centers_[:, 0], gmm.cluster_centers_[:, 1], s=20, c='red')
    # plt.title('Clusters')
    # plt.show()
    x,ys = np.meshgrid(np.sort(X[:,0]),np.sort(X[:,1]))
    XY = np.array([x.flatten(),ys.flatten()]).T
    means = gmm.means_
    covariances = gmm.covariances_
    fig = plt.figure()
    ax0 = fig.add_subplot(111)
    ax0.scatter(X[:,0],X[:,1])
    ax0.scatter(Y[0,:],Y[1,:],c='orange',zorder=10,s=100)
    for m,c in zip(means,covariances):
        multi_normal = multivariate_normal(mean=m,cov=c)
        ax0.contour(np.sort(X[:,0]),np.sort(X[:,1]),multi_normal.pdf(XY).reshape(len(X),len(X)),colors='black',alpha=0.5)
        ax0.scatter(m[0],m[1],c='red',zorder=10,s=100)
    plt.show()
  #labels = gmm.labels_
  print("\n Silhouette Score: ", silhouette_score(X, pred_y, metric='euclidean') )
#This part of the function matches between the cluster label and the original classs by identifing the maximum number of records that hav the same class in each label
#Then we change the number that represents that class with the same number that represent the label so that we will be able to calculate the kappa and the contengincy matrix
  y= pd.factorize(y)
  correcting_class= pd.DataFrame({"label":pred_y,"class":y[0]})
  lll=[]
  for i in range(5):
    lll.append(correcting_class.loc[correcting_class["class"]==i])
  for i in range(5):
    lll[i]["class"]=max(lll[i]["label"].mode())
  class_laabel_df=pd.DataFrame()
```

Figure 2.44: Expectation Maximization function.

Expectation Maximization with BOW.

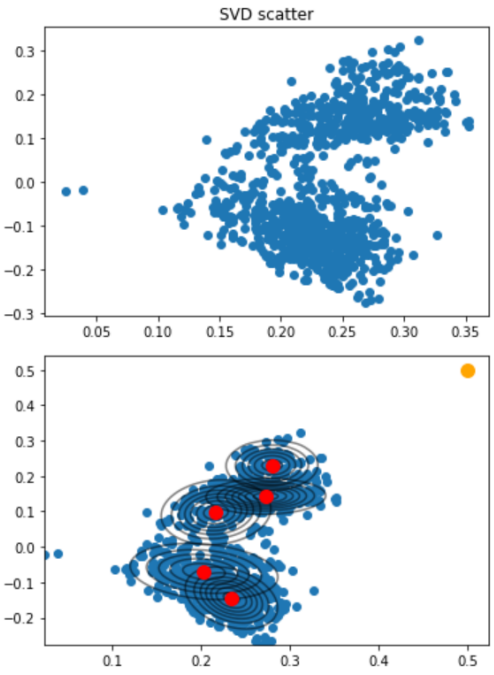
```
#em_reduction("normal", X_train_counts, y_train)
```

Figure 2.45: Expectation Maximization with BOW

The BOW transformation technique was used on the training data then was fed to the Expectation Maximization model. The model was then fitted as seen in Fig 2.45. Because there is a huge amount of inputs so it takes alot of time with out feature reduction. Therefore, we decided to perform feature reduction to enhance and improve our model.

Feature Reduction using SVD

```
em_reduction("svd", X_train_counts, y_train)
```

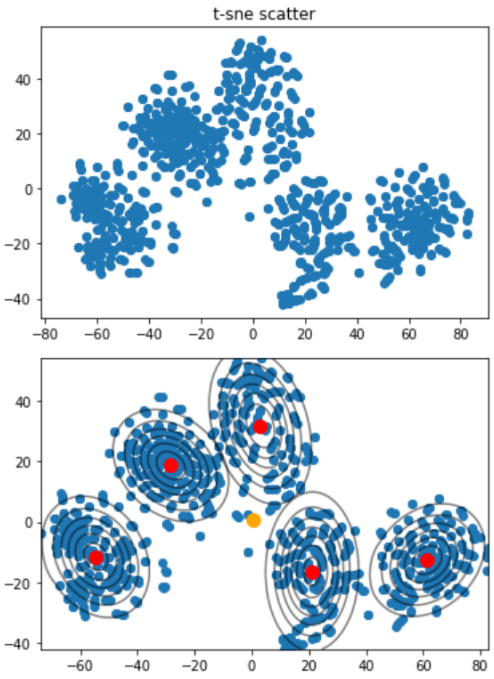



```
Silhouette Score:  0.3954964242835761

Contingency Matrix:
[[374   0  13   0 213]
[  0 141  20  26  13]
[  0  70  77  52   1]]

Kappa score:  0.4264829912847906
```

Figure 2.46: Expectation Maximization with BOW with SVD

Feature Reduction using T-SNE

```
em_reduction("tsne", X_train_counts, y_train)
```




```
Silhouette Score:  0.5206553

Contingency Matrix:
[[184   0  16   0   0]
 [  1 199   0   0   0]
 [  1   0 197   2   0]
 [  0   2   0 198   0]
 [  0   0   0   0 200]]

Kappa score:  0.9725
```

Figure 2.47: Expectation Maximization with BOW with T-SNE

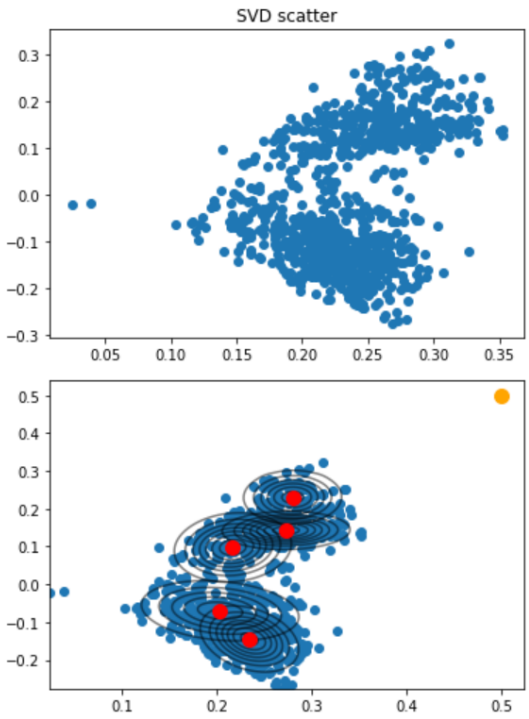
The best silhouette score and kappa in Expectation Maximization and BOW was by T-SNE, as in Fig 2.46,2.47.

Expectation Maximization with TF-IDF.

The TF-IDF transformation technique was used on the training data then was fed to the Expectation Maximization model. The model was then fitted. Because there is a huge amount of inputs so it takes alot of time with out feature reduction. Therefore, we decided to perform feature reduction to enhance and improve our model.

Feature Reduction using SVD

```
em_reduction("svd", X_train_tfidf, y_train)
```

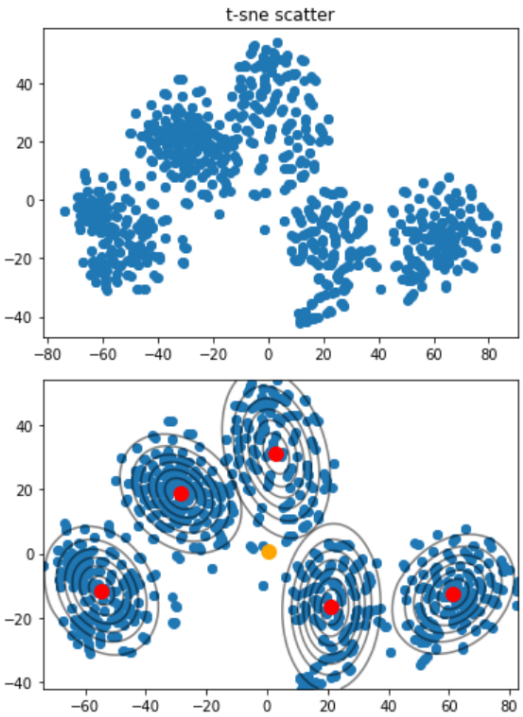



```
Silhouette Score:  0.39469154624823044

Contingency Matrix:
[[ 13 141  20   0  26]
[  1  70  77   0  52]
[217   0  13 370   0]]

Kappa score:  0.42280750910619214
```

Figure 2.48: Expectation Maximization with TF-IDF with SVD

Feature Reduction using T-SNE

```
em_reduction("tsne", X_train_tfidf, y_train)
```




```
Silhouette Score:  0.5206553

Contingency Matrix:
[[197   2   0   0   1]
[  0 198   0   2   0]
[  0   0 200   0   0]
[  0   0   0 199   1]
[ 16   0   0   0 184]]

Kappa score:  0.9725
```

Figure 2.49: Expectation Maximization with TF-IDF with T-SNE

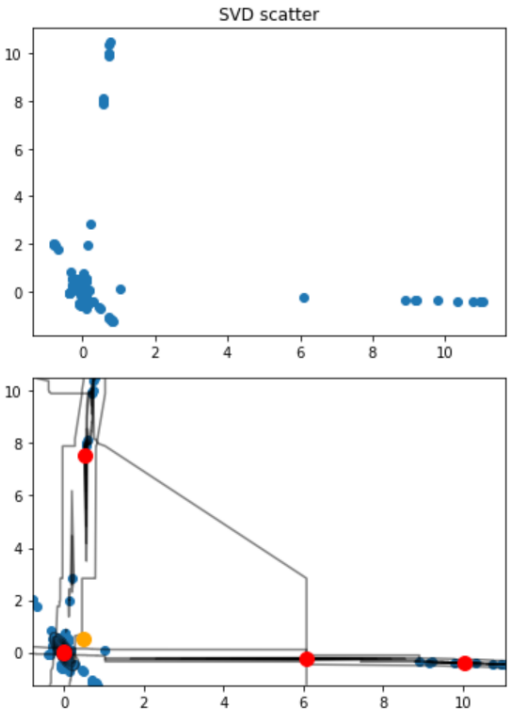
The best silhouette score and kappa in Expectation Maximization and TF-IDF was by T-SNE, as in Fig 2.48,2.49.

Expectation Maximization with n-gram.

The n-gram transformation technique was used on the training data then was fed to the Expectation Maximization model. The model was then fitted. Because there is a huge amount of inputs so it takes alot of time with out feature reduction. Therefore, we decided to perform feature reduction to enhance and improve our model.

Feature Reduction using SVD

```
em_reduction("svd", X_train_ngram, y_train)
```

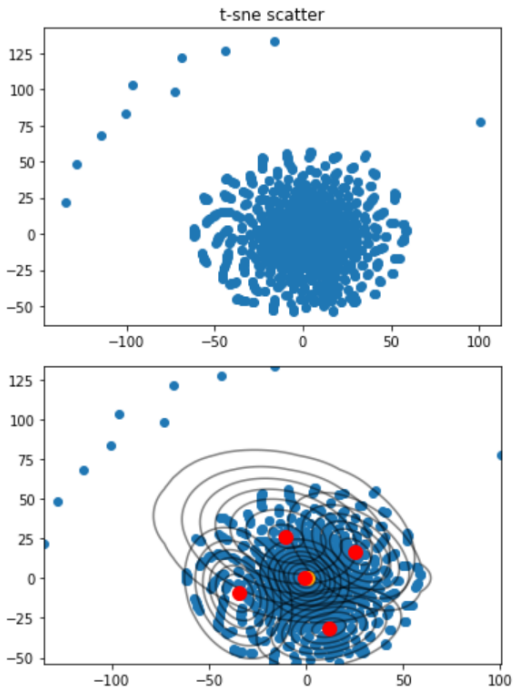



```
Silhouette Score:  0.3795924770277976

Contingency Matrix:
[[581   0   0 219   0]
[ 11   8  10 170   1]]

Kappa score:  0.4449398127507802
```

Figure 2.50: Expectation Maximization with n-gram with SVD

Feature Reduction using T-SNE

```
em_reduction("tsne", X_train_ngram, y_train)
```




```
Silhouette Score:  0.3319153

Contingency Matrix:
[[125  49   9  11   6]
[ 45 229  39  68  19]
[  0   3 138  20  39]
[  0  62   3 105  30]]

Kappa score:  0.4628099173553719
```

Figure 2.51: Expectation Maximization with n-gram with T-SNE

The best silhouette score and kappa in Expectation Maximization and n-gram was by T-SNE, as in Fig 2.50,2.51.

Expectation Maximization with LDA.

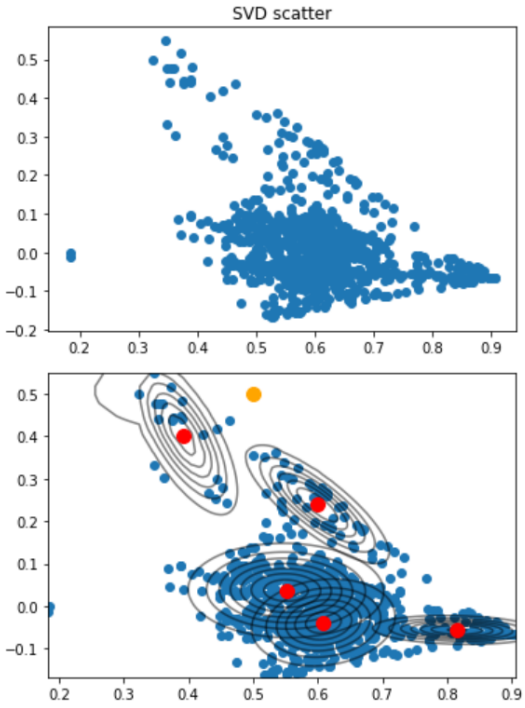
```
em_reduction("normal", X_train_LDA, y_train, 1)

Silhouette Score:  -0.008232714535958594

Contingency Matrix:
[[233  54 105 127  81]
[ 86  60  69 148  37]]

Kappa score:  0.11394217005439455
```

Figure 2.52: Expectation Maximization with LDA

The LDA transformation technique was used on the training data then was fed to the Expectation Maximization model. The model was then fitted as seen in Fig 2.52. The low kappa score indicated that we can not depend on this model with a percentage of 11 %. However, the low silhouette score indicated that the points in each cluster are far away from each other and the clusters are somewhat close to each other. Therefore, we decided to perform feature reduction to enhance and improve our model.

Feature Reduction using SVD

```
em_reduction("svd", X_train_LDA, y_train)
```

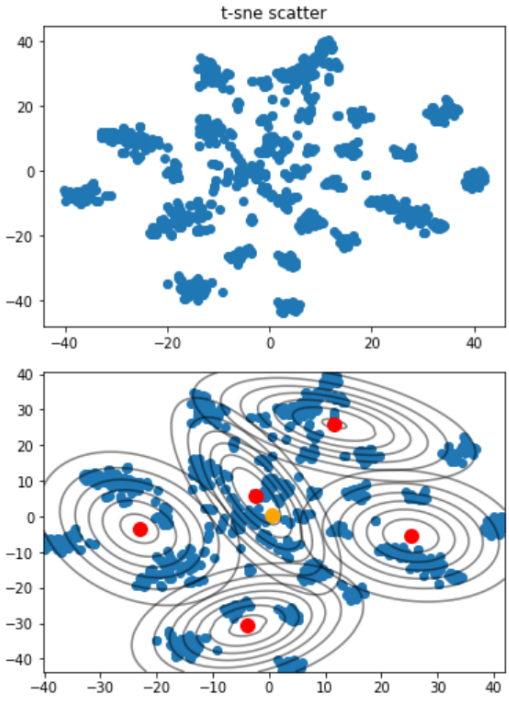



```
Silhouette Score:  0.36672002482495564

Contingency Matrix:
[[481 178  82  21  38]
[ 89  90  11   0  10]]

Kappa score:  0.1252039151712887
```

Figure 2.53: Expectation Maximization with LDA with SVD

Feature Reduction using T-SNE

```
em_reduction("tsne", X_train_LDA, y_train)
```




```
Silhouette Score:  0.36422563

Contingency Matrix:
[[132 111  70  30  57]
[100  80  98 114 208]]

Kappa score:  0.11788291900561354
```

Figure 2.54: Expectation Maximization with LDA with T-SNE

The best silhouette score and kappa in Expectation Maximization and LDA was by SVD, as in Fig 2.53,2.54.

Expectation Maximization with Doc2vec.

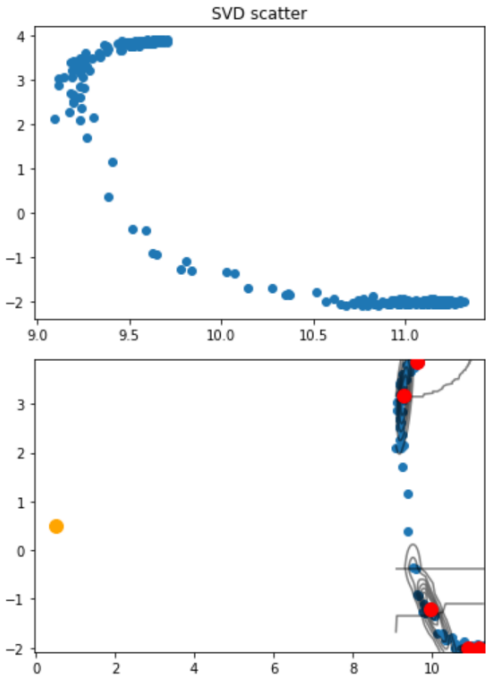
```
em_reduction("normal", X_train_D2V, y_train_D2V, 1)

Silhouette Score:  0.58596593

Contingency Matrix:
[[  9  79   5   1  51]
[ 28 165  18   9 335]]

Kappa score:  0.1671866550189276
```

Figure 2.55: Expectation Maximization with Doc2vec

The Doc2vec transformation technique was used on the training data then was fed to the Expectation Maximization model. The model was then fitted as seen in Fig 2.55. The low kappa score indicated that we can not depend on this model with a percentage of 16 %. However, the low silhouette score indicated that the points in each cluster are far away from each other and the clusters are somewhat close to each other. Therefore, we decided to perform feature reduction to enhance and improve our model.

Feature Reduction using SVD

```
em_reduction("svd", X_train_D2V, y_train_D2V)
```

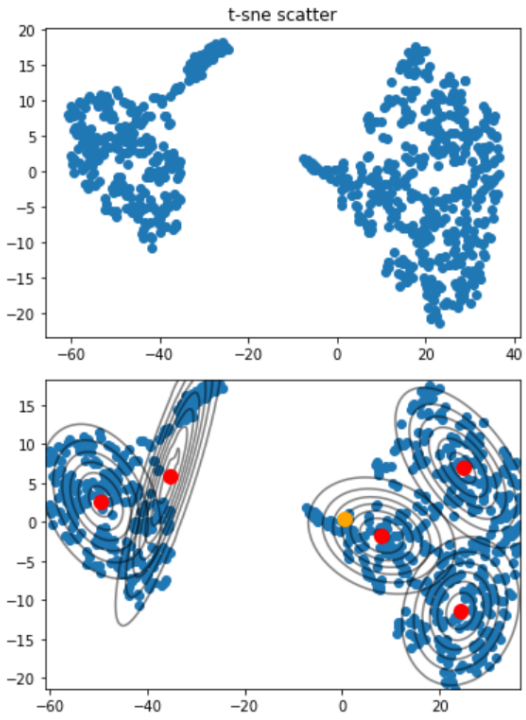



```
Silhouette Score:  0.6904043

Contingency Matrix:
[[ 69  47  15   2  12]
[146 316  37  14  42]]

Kappa score:  0.1432235001554243
```

Figure 2.56: Expectation Maximization with Doc2vec with SVD

Feature Reduction using T-SNE

```
em_reduction("tsne", X_train_D2V, y_train_D2V)
```




```
Silhouette Score:  0.4047113

Contingency Matrix:
[[47 97 54 45 45]
[27 24 33 16 29]
[40 48 69 37 89]]

Kappa score:  0.1023128707591815
```

Figure 2.57: Expectation Maximization with Doc2vec with T-SNE

The best silhouette score and kappa in Expectation Maximization and Doc2vec was by SVD, as in Fig 2.56,2.57.

2.5 Evaluation

There are many ways to evaluate a model. We chose to evaluate it using a contingency matrix, silhouette score, and Kappa. All the models produced very high Kappa and a very good contingency matrix but their silhouette score is very low. Therefore we performed feature reduction.

2.5.1 Evaluation of K-mean

Table 2.1: Evaluation of K-mean.

K-mean	SVD	T-SNE
BOW	Silhouette Score: 0.4066026738174163 Contingency Matrix: [[133 13 0 26 28] [0 103 84 13 0] [0 155 244 1 0] [61 1 0 83 55]] Kappa score: 0.4181091877496671	Silhouette Score: 0.52219236 Contingency Matrix: [[200 0 0 0 0] [0 200 0 0 0] [0 2 187 0 11] [0 2 1 197 0] [0 2 1 0 197]] Kappa score: 0.97625
TF-IDF	Silhouette Score: 0.40660267381741655 Contingency Matrix: [[133 13 0 26 28] [0 103 84 13 0] [0 155 244 1 0] [61 1 0 83 55]] Kappa score: 0.4181091877496671	Silhouette Score: 0.52219236 Contingency Matrix: [[200 0 0 0 0] [0 200 0 0 0] [0 2 187 0 11] [0 2 1 197 0] [0 2 1 0 197]] Kappa score: 0.97625
N-gram	Silhouette Score: 0.8227772755351176 Contingency Matrix: [[926 9 8 7 50]] Kappa score: 0.0	Silhouette Score: 0.3393772 Contingency Matrix: [[95 3 15 5 82] [53 188 59 70 30] [3 67 97 0 33] [18 45 8 123 6]] Kappa score: 0.3542099792099792
LDA	Silhouette Score: 0.37768751170118947 Contingency Matrix: [[284 36 69 102 109] [83 17 40 109 151]] Kappa score: 0.16395383249482087	Silhouette Score: 0.40838206 Contingency Matrix: [[63 23 54 33 27] [35 56 37 36 36] [46 21 60 50 23] [63 91 58 74 114]] Kappa score: 0.11470072627097405
Doc2vec	Silhouette Score: 0.88177454 Contingency Matrix: [[74 60 2 1 8] [157 361 10 11 16]] Kappa score: 0.16760152568992592	Silhouette Score: 0.46231106 Contingency Matrix: [[80 159 71 36 78] [69 58 50 14 85]] Kappa score: 0.09576095455009015

According to the table 2.1 the champion model for kmean is when using BOW or TF-IDF using T-SNE for feature reduction. The champion model has silhouette score of 0.522 and kappa score of 97.6 % .

2.5.2 Evaluation of Hierarchical Clustering

Table 2.2: Evaluation of Hierarchical Clustering.

Hierarchical Clustering	SVD	T-SNE
BOW	Silhouette Score: 0.3243305416837204 Contingency Matrix: [[293 0 16 91 0] [1 110 39 0 50] [17 47 73 0 63] [0 53 57 0 90]] Kappa score: 0.42592592592592593	Silhouette Score: 0.51542 Contingency Matrix: [[168 32 0 0 0] [0 198 2 0 0] [0 0 200 0 0] [1 0 0 199 0] [0 0 0 0 200]] Kappa score: 0.95625
TF-IDF	Silhouette Score: 0.324330541683 Contingency Matrix: [[293 0 16 91 0] [1 110 39 0 50] [17 47 73 0 63] [0 53 57 0 90]] Kappa score: 0.42592592592592593	Silhouette Score: 0.51542 Contingency Matrix: [[168 32 0 0 0] [0 198 2 0 0] [0 0 200 0 0] [1 0 0 199 0] [0 0 0 0 200]] Kappa score: 0.95625
N-gram	Silhouette Score: 0.81562 Contingency Matrix: [[9 921 8 7 55]] Kappa score: 0.0	Silhouette Score: 0.28112805 Contingency Matrix: [[240 112 6 23 19] [3 20 58 101 18] [73 133 3 13 178]] Kappa score: 0.36727177058668
LDA	Silhouette Score: 0.32728541212 Contingency Matrix: [[25 182 37 101 55] [47 119 64 294 76]] Kappa score: 0.1845627139744786	Silhouette Score: 0.3892 Contingency Matrix: [[172 163 253 130 82] [37 29 41 59 34]] Kappa score: 0.053645116
Doc2vec	Silhouette Score: 0.88215 Contingency Matrix: [[363 12 9 157 14] [60 3 1 74 7]] Kappa score: 0.169748353	Silhouette Score: 0.448943 Contingency Matrix: [[169 71 96 62 26] [34 32 38 19 6] [27 43 28 44 5]] Kappa score: 0.11015353602

According to the table 2.2 the champion model for Hierarchical Clustering is when using BOW or TF-IDF using T-SNE for feature reduction. The champion model has silhouette score of 0.51 and kappa score of 95 %

2.5.3 Evaluation of Expectation Maximization

Table 2.3: Evaluation of Expectation Maximization.

Expectation Maximization	SVD	T-SNE
BOW	Silhouette Score: 0.39549642428 Contingency Matrix: [[374 0 13 0 213] [0 141 20 26 13] [0 70 77 52 1]] Kappa score: 0.4264829912847906	Silhouette Score: 0.5206! Contingency Matrix: [[184 0 16 0 0] [1 199 0 0 0] [1 0 197 2 0] [0 2 0 198 0] [0 0 0 0 200]] Kappa score: 0.9725
TF-IDF	Silhouette Score: 0.3946915 Contingency Matrix: [[13 141 20 0 26] [1 70 77 0 52] [217 0 13 370 0]] Kappa score: 0.422807509106	Silhouette Score: 0.52065 Contingency Matrix: [[197 2 0 0 1] [0 198 0 2 0] [0 0 200 0 0] [0 0 0 199 1] [16 0 0 0 184]] Kappa score: 0.9725
N-gram	Silhouette Score: 0.3795924 Contingency Matrix: [[581 0 0 219 0] [11 8 10 170 1]] Kappa score: 0.444939812750	Silhouette Score: 0.33191 Contingency Matrix: [[125 49 9 11 6] [45 229 39 68 19] [0 3 138 20 39] [0 62 3 105 30]] Kappa score: 0.4628099173
LDA	Silhouette Score: 0.3667200 Contingency Matrix: [[481 178 82 21 38] [89 90 11 0 10]] Kappa score: 0.125203915171	Silhouette Score: 0.36422 Contingency Matrix: [[132 111 70 30 57] [100 80 98 114 208]] Kappa score: 0.1178829190
Doc2vec	Silhouette Score: 0.69040 Contingency Matrix: [[69 47 15 2 12] [146 316 37 14 42]] Kappa score: 0.1432235001	Silhouette Score: 0.404 Contingency Matrix: [[47 97 54 45 45] [27 24 33 16 29] [40 48 69 37 89]] Kappa score: 0.102312876

According to the table 2.3 the champion model for Expectation Maximization is when using BOW or TF-IDF using T-SNE for feature reduction. The champion model has silhouette score of 0.52 and kappa score of 97.2 %

Now we have for each combination between (model, transformation and feature reduction) the best Kappa score with regards the contingency matrix and silhouette score. The TF-IDF and BOW transformation produced the best performance for each model using T-SNE feature reduction. Comparing all the performances of the models, the K-mean model outperformed them all with the highest Kappa of 97.6%, silhouette of 0.522 and comparing the output clusters with the label using contingency matrix. The diagonal of contingency matrix represents the records that where labeled correctly, while the other values represent the mislabel records and we could see that the clustering model labels the records near to the human label.

2.5.4 Error Analysis

Analyzing the error for the clustering models was a little bit challenging because we don't know every cluster corresponds to which class. So, first we had to identify every cluster and match it to the right class. For each record we had the right class and the cluster label. We could identify the class of the cluster by tracking the records of the same class that grouped together in one cluster. The output was like the corresponding data frame, as in Fig 2.58.

	label name	sample	label	class
0	The Meaning of Truth	oppon might say intellect kind slush make repl...	0	0
1	The Meaning of Truth	function grow wish account shall true whether ...	0	0
2	The Meaning of Truth	india sit exactli mean say know tiger precis f...	0	0
3	The Meaning of Truth	beyond sound know realiti faintest fragmentari...	0	0

Figure 2.58: The output of the cluster and labels dataframe

After this we grouped all the records for which the class is different from the cluster label for further analysis as in the following figure 2.59.

	label name	sample	predicted_class	class
61	The Meaning of Truth	qui rendu patent et visibl le long travail sou...	2	0
873	Indiana	decay besid curiou shapeless pagoda spot rende...	2	1
820	Indiana	think concern suprem felic fellowman think eve...	3	1
839	Indiana	societi individu simpli regard opinion freeli ...	3	1
652	Notes from the Underground	stone step salt fish sort hand cri wail someth...	1	2
650	Notes from the Underground	promot let make digress russian speak gener ne...	1	2
695	Notes from the Underground	think pass need litti idyl affectedli bookishl...	1	2
676	Notes from the Underground	noth would come phase sceptic indiffer everyth...	1	2
674	Notes from the Underground	almost nudg begin stop understand quiver somet...	1	2
621	Notes from the Underground	liza man fond reckon troubl count joy count ou...	1	2
617	Notes from the Underground	anoth circumst worri day one like unlik anyon ...	1	2
615	Notes from the Underground	like book make even outsid feel sick though lo...	1	2
642	Notes from the Underground	duti fail draw father mother nearer peopl say ...	1	2
627	Notes from the Underground	perhap anoth thing liza man fond reckon troubl...	1	2
770	Notes from the Underground	love holi mysteri ought hide eye whatev happen...	1	2
759	Notes from the Underground	talk like truli must unlucki h sort thing most...	1	2
756	Notes from the Underground	afterward lay futur children grow feel exampl ...	1	2
753	Notes from the Underground	wail someth luck beat fish step cabmen drunken...	1	2
785	Notes from the Underground	piti woman know wrong heart fail suffer love l...	1	2
703	Notes from the Underground	would torment make feel know may torment man p...	1	2
219	A Discourse on Method	must rule investig resolv commenc therefor exa...	0	3
236	A Discourse on Method	intent account attempt master particular scien...	0	3

Figure 2.59: The output of the cluster that not equal label in dataframe

After this we grouped all records of this table with the same class to identify the most frequent words, as in Fig 2.66,2.67.

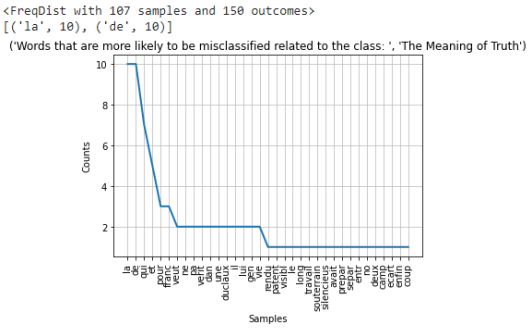


Figure 2.60: the most frequent word.

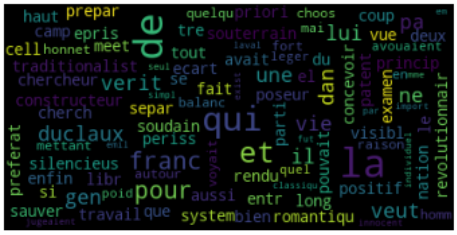


Figure 2.61: visualization of the words

For this class only one record was mislabeled, the record is a French quotation that is why the algorithm couldn't identify it as part of the cluster.(It can be considered as an outlier)

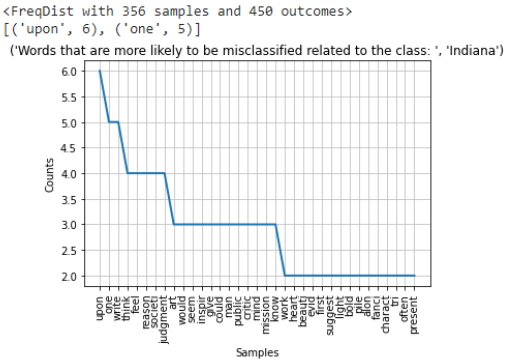


Figure 2.62: the most frequent word.

Figure 2.63: visualization of the words

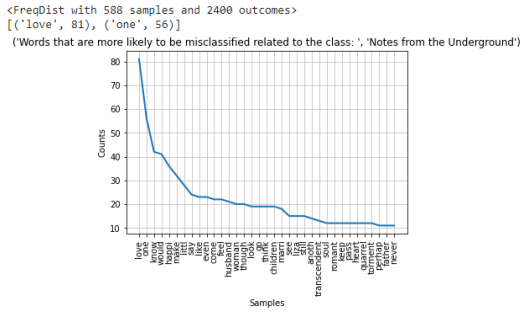


Figure 2.64: the most frequent word.

Figure 2.65: visualization of the words

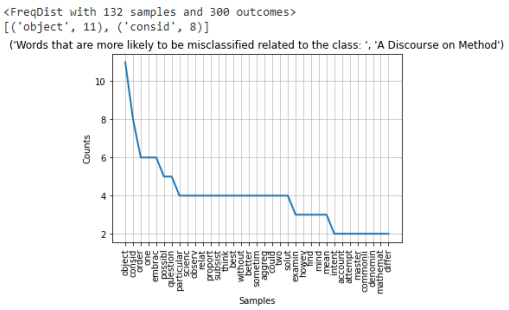


Figure 2.66: the most frequent word.

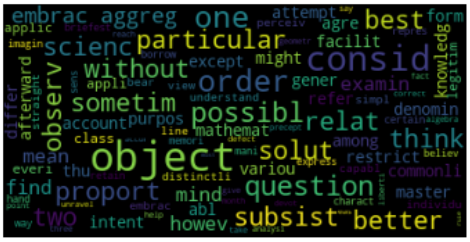


Figure 2.67: visualization of the words

As we can see from the previous figures that the records that were miss-labeled contained common words like ("one, write, think, ...ect)

2.6 Future Work

In the future, we can generalize this model to include more philosophical schools (labels). we will work on changing the hyper parameters to produce better models with better kappa and silhouette. we can use different models like a deep learning model. We can use different combinations of books and increase the number of partitions.

Chapter 3

Conclusions

In conclusion, the K-mean model had the best performance with the Expectation Maximization coming close. The TF-IDF and BOW transformations turned out to be the best transformation along with reducing the corresponds with feature vector using T-SNE algorithm for this kind of problem which is in text clustering. In the error analysis we concluded that the some samples were wrongly grouped because the words in the samples were also frequently found in other books.

References

[1] F. Beil, M. Ester, and X. Xu, "Frequent term-based text clustering," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002, pp. 436–442.