GNG5125 Data Science Applications

Group Assignment-II: Text Clustering

University of Ottawa

Faculty of Engineering

Guided By:

Dr. Arya Rahgozar

Presented by:

Group 7:

Leenanci Parmar
Prashant Kaushik
Vikram Khanzode

Date: 3rd March 2022

1. Introduction

Clustering is the technique of grouping a set of objects in such a way that objects in the same group (cluster) are, in some sense, more similar to each other than to those in other group. It is widely used in many fields, including pattern recognition, information retrieval, data compression, computer graphics and machine learning. Clustering serves as an unsupervised machine learning algorithm wherein data points are grouped into different clusters, consisting of similar data points. It does it by finding some similar patterns in the unlabelled dataset.

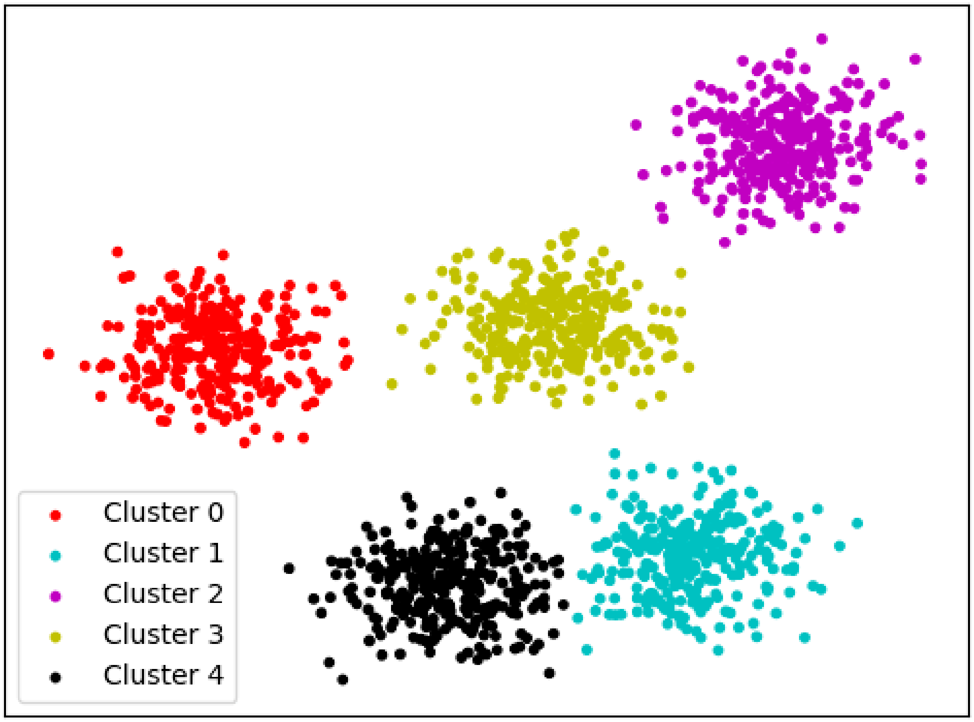


Figure 1: Clustering [4]

Why do we need clustering?

Labeled data is expensive. With clustering, we can use unlabelled data to solve machine learning problems. It is widely used in tasks such as Marketing analysts where we can can separate data by gender, interests, income, places visited etc.Clustering has enormous applicability and we can apply clustering to solve a myriad of problems. The clustering methods are broadly divided into **Hard clustering** (datapoints belong to only one group) and **Soft Clustering** (data points can belong to another group also). But there are also other various approaches of Clustering exist.

In this project, we have used Partitioning (K-means), Hierarchical (Agglomerative) and Distribution Model-Based (Expectation-Maximization) algorithms on Gutenberg's books samples.

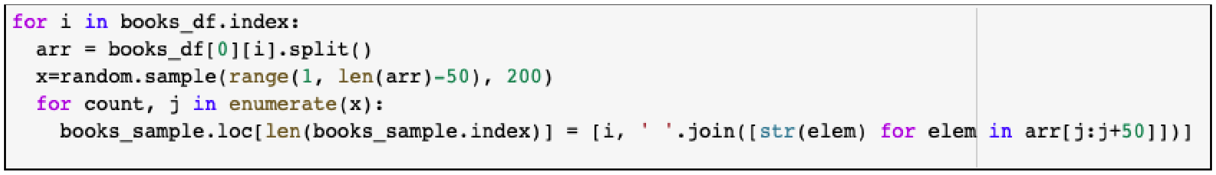
2. Dataset

For this assignment, we have taken five different samples of Gutenberg digital books, which are of five different authors and we have preprocessed these samples into a dataframe titled as “books_data.csv”. The names of the books from horror and one related to birds are as below:

1) Emma by Jane Austen
2) Poems of William Blake by William Blake
3) Stories to Tell to Children by Sara Cone Bryant
4) Alice’s Adventures in Wonderland, by Lewis Carroll
5) Moby Dick; Or, The Whale by Herman Melville

3. Data Preparation

With the use of the NLTK library, we read the books and then created random 200 samples of each book which contains 150 words. Each random sample is labelled by the author’s name. Figure 2 illustrates the code to prepare, for example, 200 samples of 50 words each.



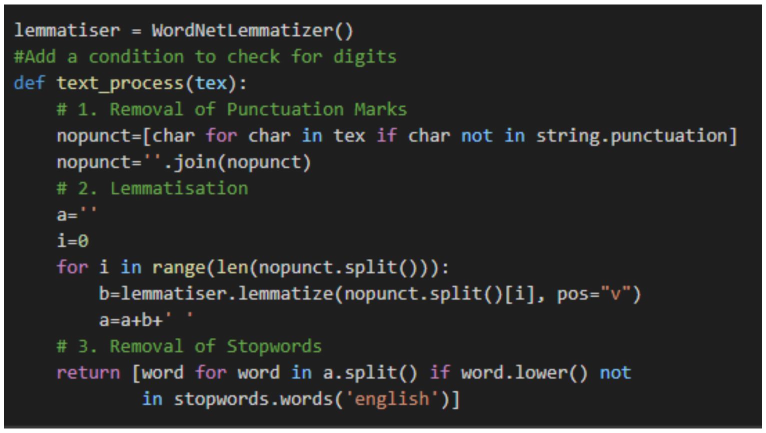
```
for i in books_df.index:
  arr = books_df[0][i].split()
  x=random.sample(range(1, len(arr)-50), 200)
  for count, j in enumerate(x):
    books_sample.loc[len(books_sample.index)] = [i, ' '.join([str(elem) for elem in arr[j:j+50]])]
```

Figure 2: Data Preparation.

4. Preprocessing

Data preprocessing is the procedure for preparing raw data for use in a machine learning model. It's the first and most important stage in building a machine learning model. For this project, we have removed the stop words and garbage characters.

In addition we’ve performed lemmatization to convert words into their lemma form. Figure 3 shows the implementation to remove stop words and perform lemmatization.



```
lemmatiser = WordNetLemmatizer()
#Add a condition to check for digits
def text_process(tex):
    # 1. Removal of Punctuation Marks
    nopunct=[char for char in tex if char not in string.punctuation]
    nopunct=''.join(nopunct)
    # 2. Lemmatisation
    a=''
    i=0
    for i in range(len(nopunct.split())):
        b=lemmatiser.lemmatize(nopunct.split()[i], pos="v")
        a=a+b+' '
    # 3. Removal of Stopwords
    return [word for word in a.split() if word.lower() not
            in stopwords.words('english')]
```

Figure 3: Removal of stop words & Lemmatization

5. Feature Engineering

The process of turning raw data into a format or structure that is more suitable for model building and data discovery, in general, is known as data transformation. It's a crucial phase in feature engineering that makes finding insights easier. For this project, we have transformed the pre-processed data into BOW, TF-IDF and BERT. These are all methods for converting text phrases to numeric vectors.

5.1 BOW

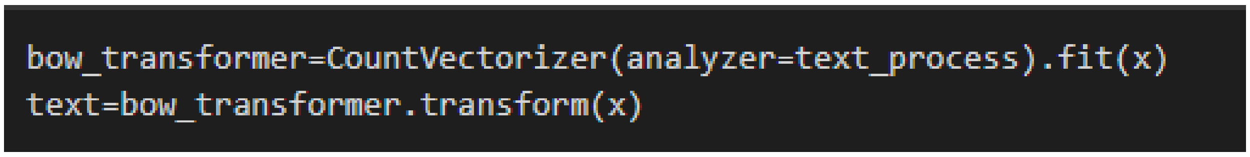
The Bag of Words (BoW) model is the most basic type of numerical text representation. A phrase can be represented as a bag of words vector, just like the term itself (a string of numbers).

For example, BOW for below two sentences is as follow:
Sentence 1: It is spooky and good
Sentence 2: It is not good and is slow
We will first build a vocabulary from all the unique words in the above two sentences. The vocabulary consists of these 7 words: ‘It’, ‘is’, ‘spooky’, ‘and’, ‘good’, ‘not’, ‘slow’.

We can now take each of these words and mark their occurrence in the two sentences above with 1s and 0s. This will give us 2 vectors for 2 sentences:

	1 It	2 is	3 spooky	4 and	5 good	6 not	7 slow	Length of sentence (in words)
Sentence 1	1	1	1	1	1	0	0	5
Sentence 2	1	2	0	1	1	1	1	7

Vector of Sentence 1: [1 1 1 1 1 0 0]
Vector of Sentence 2: [1 2 0 1 1 1 1]



```
bow_transformer=CountVectorizer(analyzer=text_process).fit(x)
text=bow_transformer.transform(x)
```

Figure 4. BOW implementation

5.2 TF-IDF

A numerical metric called term frequency-inverse document frequency is meant to show how essential a word is to a document in a collection or corpus.

Term frequency-The frequency of a term t mentioned in document d is measured by TF:

$$tf_{t,d} = \frac{n_{t,d}}{Number\ of\ terms\ in\ the\ document}$$

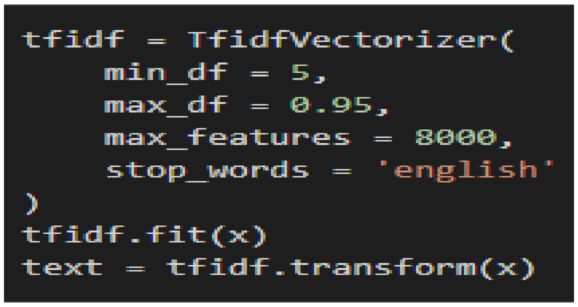
Here, in the numerator, n is the number of times the term “t” appears in the document “d”. Thus, each document and term has its own TF value.

Inverse Document Frequency-IDF is how important a term is. Computing TF is not sufficient to understand the significance of words. Therefore, IDF is also necessary.

$$idf_t = \log \frac{number\ of\ documents}{number\ of\ documents\ with\ term\ 't'}$$

TF-IDF is computed as below:

$$(tf\_idf)_{t,d} = tf_{t,d} * idf_t$$



```
tfidf = TfidfVectorizer(
    min_df = 5,
    max_df = 0.95,
    max_features = 8000,
    stop_words = 'english'
)
tfidf.fit(x)
text = tfidf.transform(x)
```

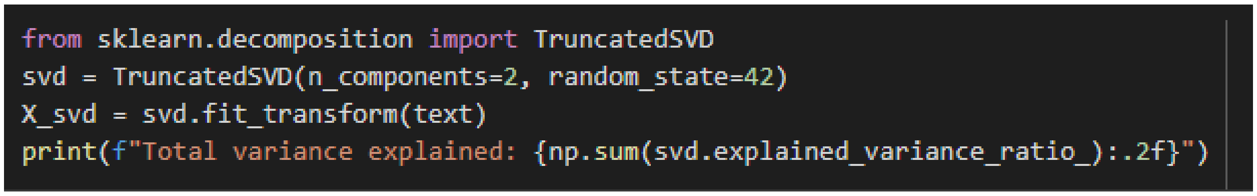
Figure 5.TF-IDF implementation

5.3 BERT

In this project we have also used BERT to transform our data into a more desirable form. Bidirectional Encoder Representations from Transformers (BERT) is a transformer-based machine learning technique for natural language processing (NLP) pre-training developed by Google. BERT was created and published in 2018 by Jacob Devlin and his colleagues from Google. In 2019, Google announced that it had begun leveraging BERT in its search engine, and by late 2020 it was using BERT in almost every English-language query [6][7].

6. Dimensionality Reduction

Dimensionality reduction is the transformation of data from a high-dimensional space into a low-dimensional space so that the low-dimensional representation retains some meaningful properties of the original data, ideally close to its intrinsic dimension. Working in high-dimensional spaces can be undesirable for many reasons; raw data are often sparse as a consequence of the curse of dimensionality, and analyzing the data is usually computationally intractable (hard to control or deal with) [8]. For this project we have used Truncated Singular Value Decomposition (SVD) to simplify our modeling.



```
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components=2, random_state=42)
X_svd = svd.fit_transform(text)
print(f"Total variance explained: {np.sum(svd.explained_variance_ratio_):.2f}")
```

Figure 6: Implementation of Dimensionality Reduction using SVD

7. Clustering Models

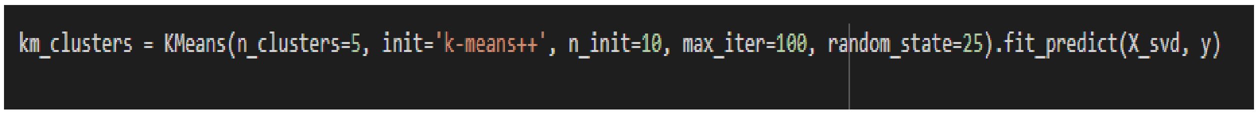
6.1 K-means clustering

K-Means Clustering is an iterative Unsupervised Learning algorithm, which groups the unlabelled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on. It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.
We first select the number of clusters K and select centroids. Then we assign each data point to their closest centroid to form predefined K clusters. Then we calculate the variance and create a new centroid of each cluster and then iteratively reassign each datapoint to the new closest centroid of each cluster and stop when there is no reassignment.

To find K, we use elbow method which works on the concept of WCSS value. WCSS stands for Within Cluster Sum of Square Distances which is the sum of squared distances between each point and the centroid in a cluster. We calculate WCSS value for each cluster as follows:

$$\boxed{\begin{array}{c} \mathrm{WCSS}(k) = \sum_{j=1}^{k} \sum_{\mathbf{x}_i \in \text{cluster } j} \|\mathbf{x}_i - \bar{\mathbf{x}}_j\|^2, \\ \text{where } \bar{\mathbf{x}}_j \text{ is the sample mean in cluster } j \end{array}}$$

We execute the K-means clustering on a given dataset for different K values and calculate the WCSS value. We then plot a curve between calculated WCSS values and the number of clusters K. The sharp point of bend in the graph is considered as the best value of K.



```
km_clusters = KMeans(n_clusters=5, init='k-means++', n_init=10, max_iter=100, random_state=25).fit_predict(X_svd, y)
```

Figure 7: Implementation of Kmeans Clustering

TF-IDF: The following are the SSE plot, SVD Cluster plot for Kmeans clustering algorithm for the dataset transformed through TFIDF.

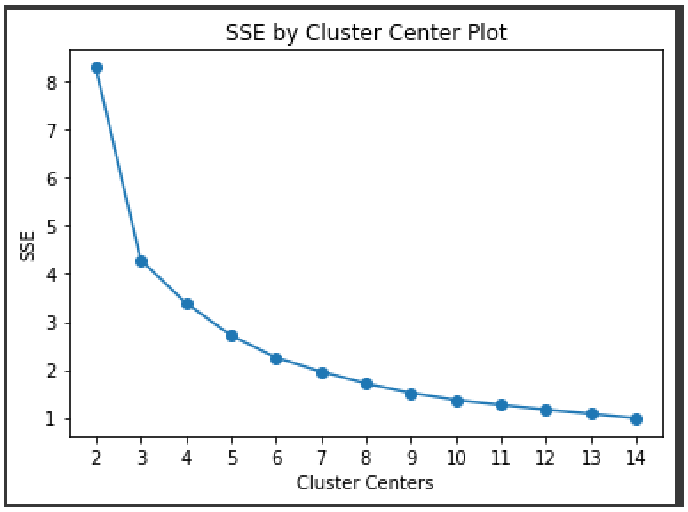


Figure 8: Kmeans clustering SSE plot for TFIDF transformed data

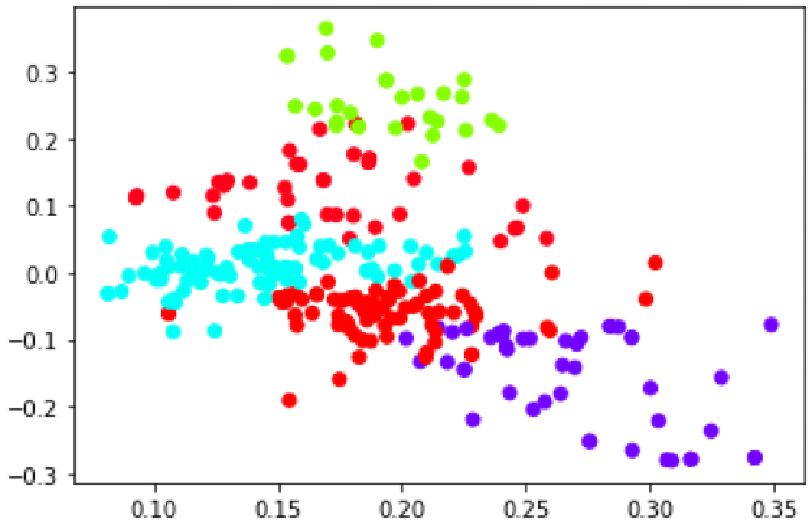


Figure 9: SVD Cluster Plot for Kmeans clustering algorithm on TFIDF transformed data

BOW: The following are the SSE plot, SVD Cluster plot for Kmeans clustering algorithm for the dataset transformed through BOW.

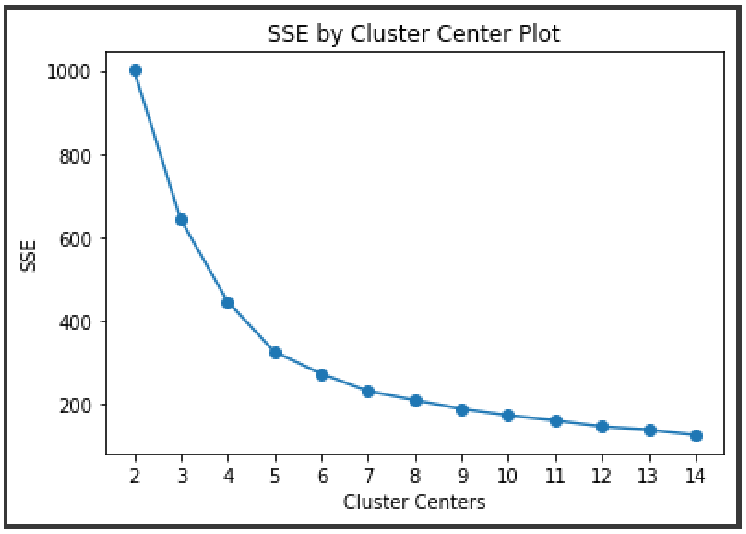


Figure 10: Kmeans clustering SSE plot for BOW transformed data

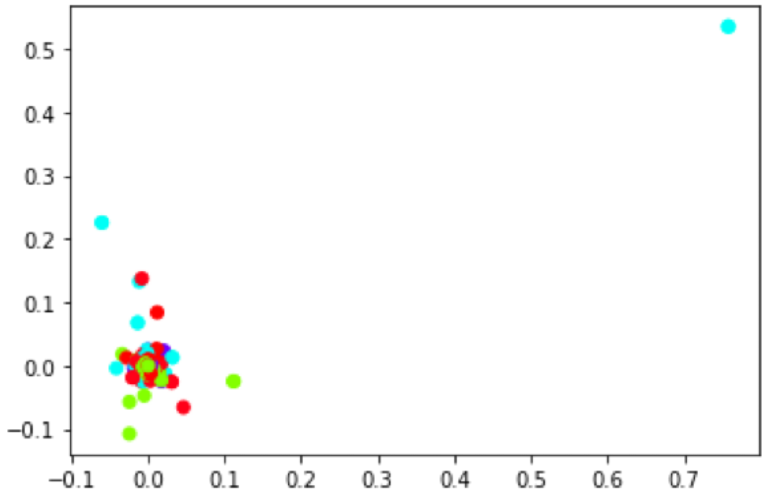


Figure 11: SVD Cluster Plot for Kmeans clustering algorithm on BOW transformed data

BERT: The following are the SSE plot, SVD Cluster plot for Kmeans clustering algorithm for the dataset transformed through BERT.

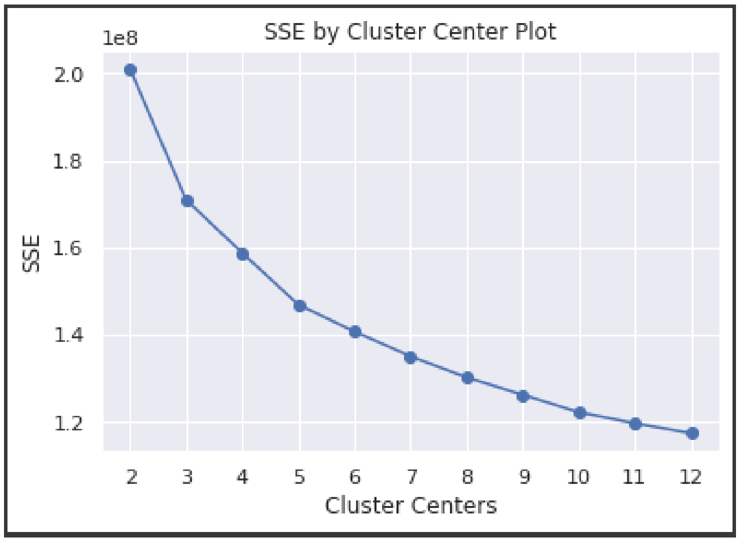


Figure 12: Kmeans clustering SSE plot for BERT transformed data

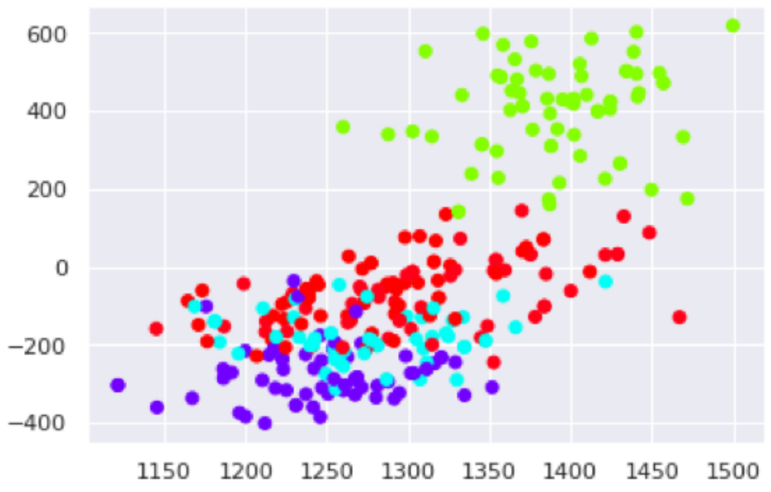


Figure 13: SVD Cluster Plot for Kmeans clustering algorithm on BERT transformed data

6.2 Hierarchical Clustering

Hierarchical clustering is a unsupervised machine learning algorithm, which is used to group the unlabelled datasets into a cluster. First, we consider all the points as a single cluster. Then we merge two closest data points or clusters to form one cluster. We repeat this step until only one cluster left. The dendrogram is a tree-like structure that is used to store each step as a memory that the HC algorithm performs. In the dendrogram plot, the Y-axis shows the Euclidean distances between the data points, and the x-axis shows all the data points of the given dataset. We can use Dendrogram to find the optimal number of clusters for clustering. We find the maximum vertical distance that does not cut any horizontal bar.

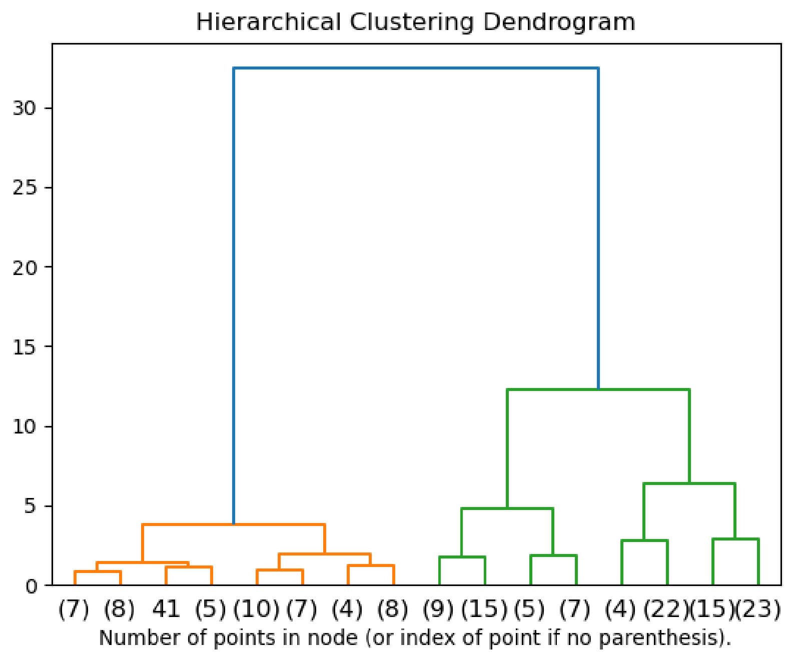
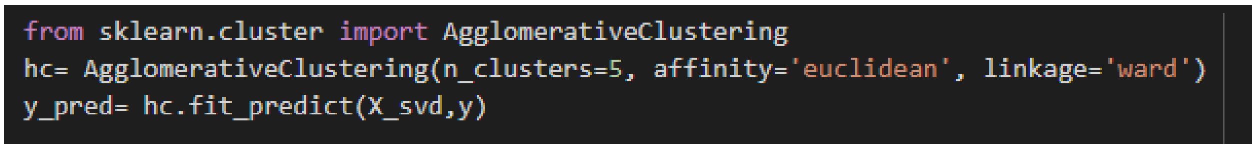


Figure 14: An example of Heirarchical Clustering Dendogram



```
from sklearn.cluster import AgglomerativeClustering
hc= AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')
y_pred= hc.fit_predict(X_svd,y)
```

Figure 15: Implementation of Heirarchical Clustering Algorithm

TF-IDF: The following are the SVD Cluster plot for Heirarchical clustering algorithm for the dataset transformed through TFIDF:

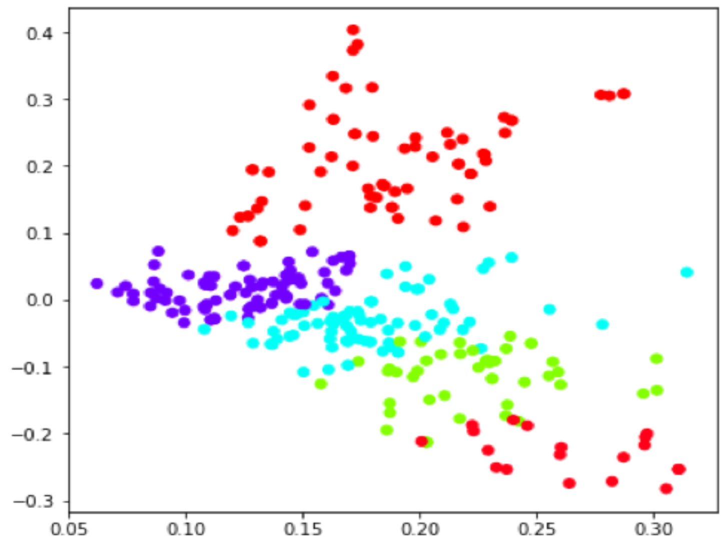


Figure 16: SVD Cluster Plot for Heirarchical clustering algorithm on TFIDF transformed data

BOW: The following are the SVD Cluster plot for Heirarchical clustering algorithm for the dataset transformed through BOW:

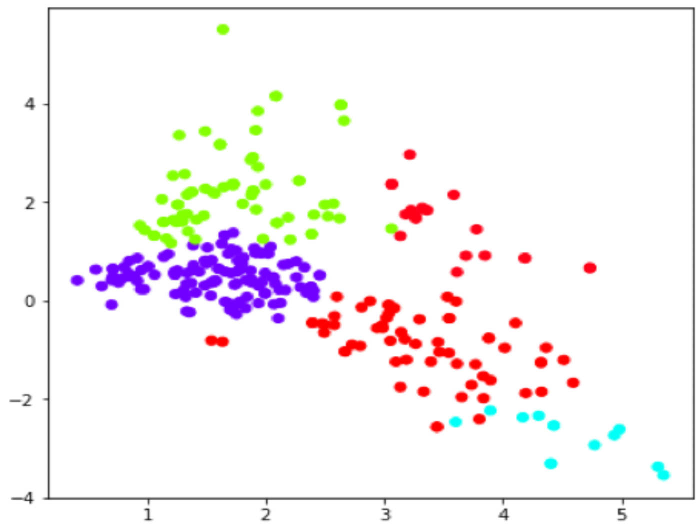


Figure 17: SVD Cluster Plot for Heirarchical clustering algorithm on BOW transformed data

BERT: The following are the SVD Cluster plot for Heirarchical clustering algorithm for the dataset transformed through BERT:

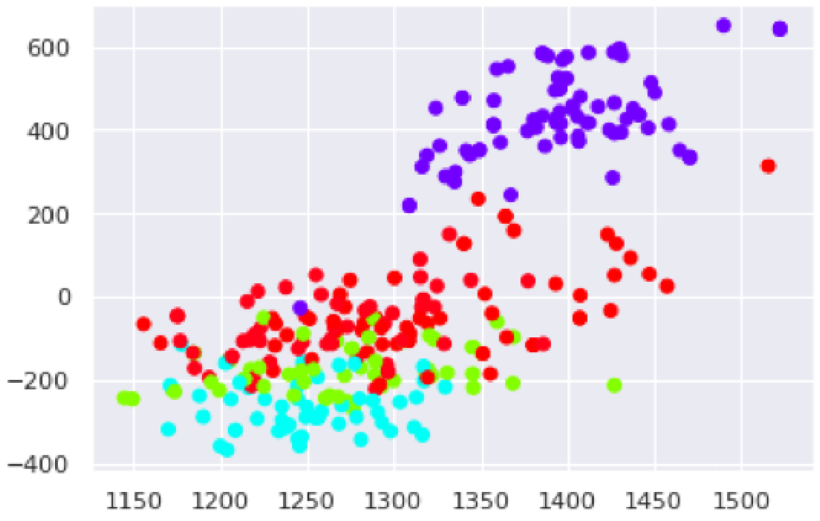
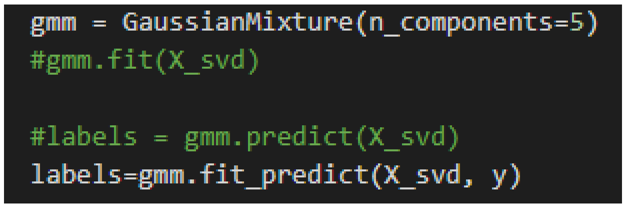


Figure 18: SVD Cluster Plot for Heirarchical clustering algorithm on BERT transformed data

6.3 Expectation Maximization (EM) Algorithm

The third type of clustering model that we used for our project was the EM model. The expectation-maximization (EM) algorithm is an approach for performing maximum likelihood estimation in the presence of latent variables. It does this by first estimating the values for the latent variables, then optimizing the model, then repeating these two steps until convergence. It is an effective and general approach and is most commonly used for density estimation with missing data, such as clustering algorithms like the Gaussian Mixture Model. [9]



```
gmm = GaussianMixture(n_components=5)
#gmm.fit(X_svd)

#labels = gmm.predict(X_svd)
labels=gmm.fit_predict(X_svd, y)
```

Figure 19: Implementation of Expectation Maximization (EM) algorithm

TF-IDF: The following is the Gaussian Mixture Scatter Plot for EM algorithm on TFIDF transformed data:

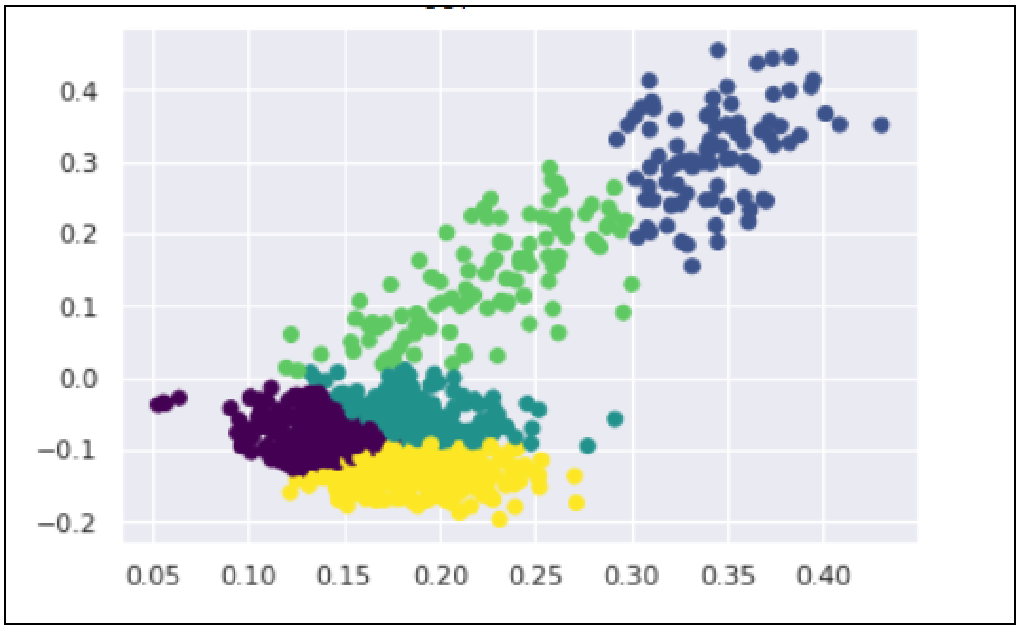


Figure 20: Gaussian Mixture Scatter Plot for EM algorithm on TFIDF transformed data

BOW: The following is the Gaussian Mixture Scatter Plot for EM algorithm on BOW transformed data:

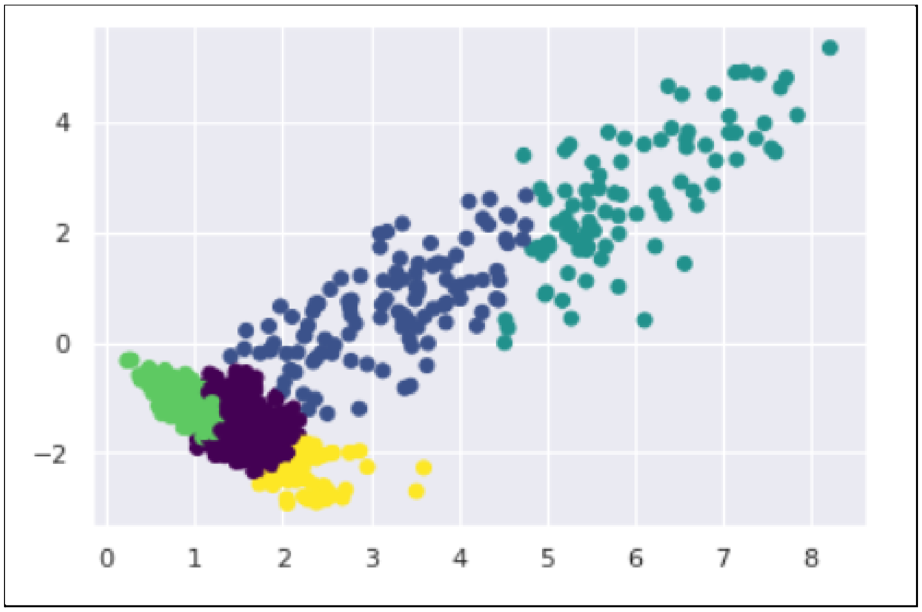


Figure 21: Gaussian Mixture Scatter Plot for EM algorithm on BOW transformed data

BERT: The following is the Gaussian Mixture Scatter Plot for EM algorithm on BERT transformed data:

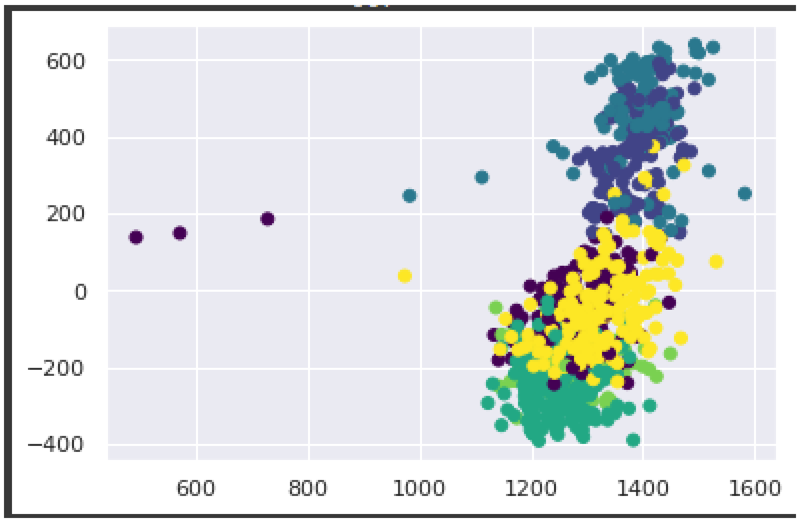


Figure 22: Gaussian Mixture Scatter Plot for EM algorithm on BERT transformed data

8. Error Analysis & Comparison

8.l Silhoutte Scores:

Silhouette Coefficient or silhouette score is a metric used to calculate the goodness of a clustering technique. Its value ranges from -1 to 1.

1: Corresponds to clusters being well apart from each other and clearly distinguished.

0: Corresponds to clusters being indifferent, or we can say that the distance between clusters is not significant.

-1: Corresponds to clusters being assigned in the wrong way. [10]

	k=2	k=3	k=4	k=5	k=6
Kmeans + TFIDF	0.5593	0.4871	0.4050	0.3869	0.3960
Kmeans + BOW	0.4811	0.4621	0.4362	0.4428	0.4414
Kmeans + BERT	0.2351	0.2080	0.1981	0.1970	0.1837
Hierarchical + TFIDF	0.5487	0.5127	0.3737	0.3760	0.3498
Hierarchical + BOW	0.5288	0.4709	0.3775	0.3959	0.3772
Hierarchical + BERT	0.3707	0.3552	0.3964	0.2948	0.2857
EM + TFIDF	0.7269	0.3394	0.3872	0.3644	0.3707
EM + BOW	0.3701	0.3682	0.3639	0.3708	0.3682
EM + BERT	0.2338	0.2038	0.1943	0.1939	0.1780

Table 1: Silhouette Scores for various values of k

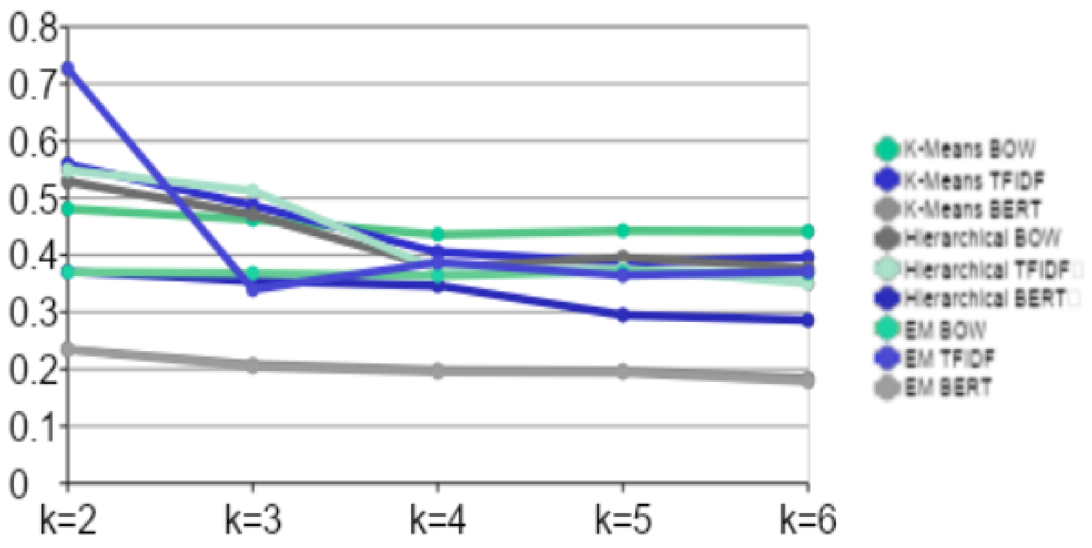


Figure 23: Silhouette Scores for various values of k

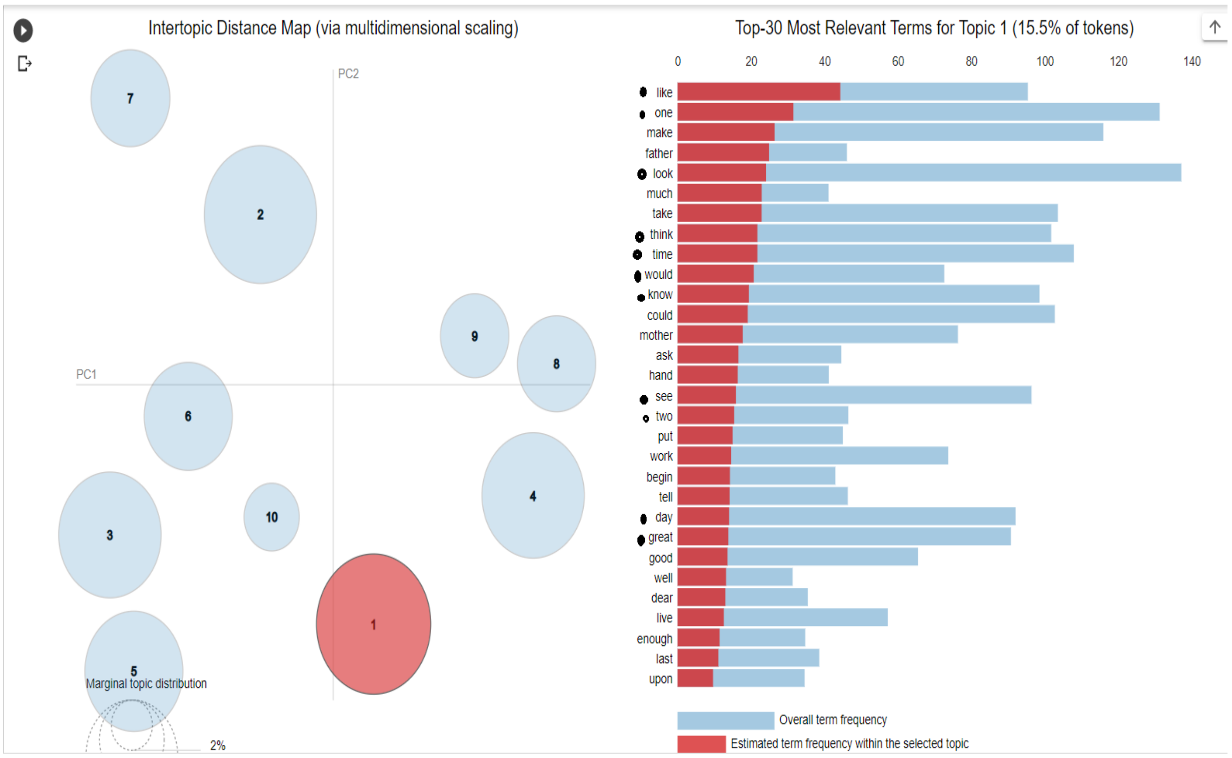
8.2 Cohen-Kappa Score

	Cohen-Kappa Score for k=5
KMeans Clustering with TFIDF	0.1687
Kmeans Clustering with BOW	0.0112
Kmeans Clustering with BERT	0.425
Hierarchical Clustering with TFIDF	-0.1112
Hierarchical Clustering with BOW	0.0812
Hierarchical Clustering with BERT	0.0187
EM Clustering with TFIDF	0.3707
EM Clustering with BOW	0.2725
EM Clustering with BERT	0.515

Table 2: Cohen Kappa scores for all the implementations of clustering models

8.3 Latent DA

In natural language processing, 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. For example, if observations are words collected into documents, it posits that each document is a mixture of a small number of topics and that each word's presence is attributable to one of the document's topics. LDA is an example of a topic model. We have implemented LDA in this projects for labels 2 & 4 since they have similar patterns of words.



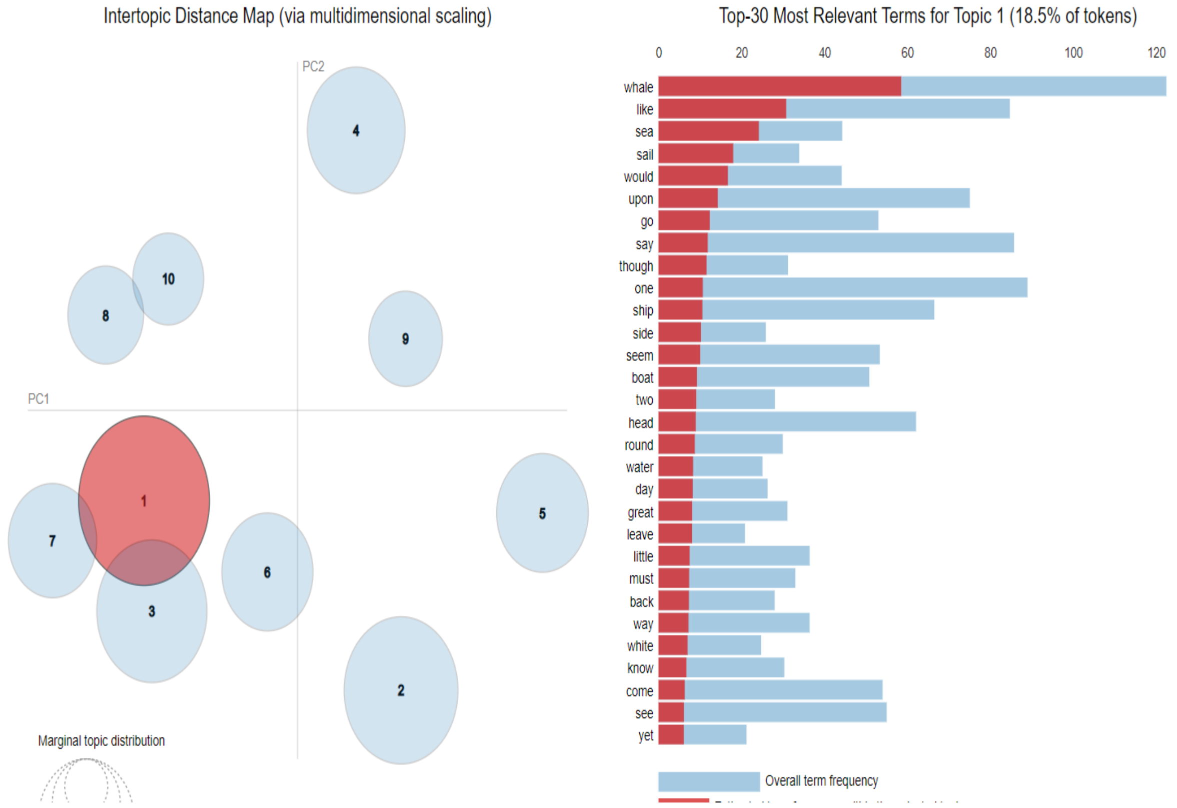


Figure 24: LDA implementation for labels 2 & 4

9. Conclusions

- EM using BERT is the champion model.
- BOW in general generated bad results and K-means clustering using BOW gave the least Kappa.
- Reducing the dimensionality improved every algorithm.
- Books 0 was easiest but all the models confused between books 2 and 4.
- Generating word embeddings using BERT is a computationally expensive (35-40 minutes)

References:

1) JavaPoint “Clustering in Machine Learning”:url
2) JavaPoint “K-Means Clustering Algorithm”: url
3) JavaPoint “Heirarchical CLustering in Machine Learning”: url
4) Issarane, Hausmane. “Clustering: Definition”: url
5) Devlin, Jacob; Chang, Ming-Wei; Lee, Kenton; Toutanova, Kristina (11 October 2018). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". arXiv:1810.04805v2 [cs.CL].
6) "Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing". *Google AI Blog*. Retrieved 2019-11-27.
7) Rogers, Anna; Kovaleva, Olga; Rumshisky, Anna (2020). "A Primer in BERTology: What We Know About How BERT Works". *Transactions of the Association for Computational Linguistics*. **8**: 842–866. arXiv:2002.12327. doi:10.1162/tacl_a_00349. S2CID 211532403.
8) van der Maaten, Laurens; Postma, Eric; van den Herik, Jaap (October 26, 2009). "Dimensionality Reduction: A Comparative Review" (PDF). *J Mach Learn Res*. **10**: 66–71.
9) Brownlee, Jason. (2019, November 1) “A Gentle Introduction to Expectation-Maximization (EM) Algorithm” Machine Learning Mastery url
10) Bharadwaj, Ashutosh. (2020, May 26) “Silhouette Coefficient: Validating clustering techniques” Towards Data Sciecne: url