CS6370 - Natural Language Processing
Improvements on Vector Space Model

<u>Team-36</u>

1. Palash Behra- CS22M061

2. Ravi Singh - CS22M069

1. Introduction

An information retrieval (IR) system is a software system that retrieves relevant information from a large collection of unstructured or semi-structured data in response to a user's query. It involves a process of analyzing, indexing, and storing the data in a way that allows for efficient retrieval of information that is relevant to the user's query. The goal of an IR system is to provide users with a ranked list of documents or information items that are relevant to their query, based on their search criteria. IR systems can be applied in various domains, including web search engines, digital libraries, and enterprise search systems. The effectiveness of an IR system is measured using various metrics such as precision, recall, and F1-score, which evaluate the relevance of the retrieved documents to the user's query. The design and implementation of an effective IR system involve various techniques and methods from fields such as natural language processing, machine learning, and data mining.

The information retrieval **vector space model** (VSM) is a prominent method for representing and ranking documents based on their similarity to a query. While it offers some benefits, such as simplicity, efficiency, and flexibility, it also has certain disadvantages.

1.1. Limitations

● Sparsity: The VSM is based on a sparse representation of documents and queries as high-dimensional vectors, with the majority of elements being 0 or near-zero. Its sparsity can make evaluating similarity and relevance challenging, especially when dealing with brief or noisy searches.

● Term weighting: Term weighting systems are used by the VSM to alter the relevance of terms in documents and queries. Unfortunately, selecting the appropriate word weighting system and parameters is not always easy, and different schemes might result in different ranks and outcomes.

● The Vector Space Model (VSM) treats terms as independent units without taking into consideration their semantic relationships or contextual relevance. Consequently, this approach may fail to capture the intended meaning or objective of a query, leading to incomplete or irrelevant results. Moreover, the original sequence of terms in a document is disregarded in the vector space representation, making it unsuitable for modeling sequences of terms in documents.

● Difficulties in dealing with synonyms and homonyms: The VSM recognizes various concepts as distinct, even if they have similar (synonymous) or dissimilar (homonymous) meanings (homonyms). This might result in mismatches between the query and related documents, reducing the retrieval system's overall efficacy.

● As a part of the assignment, we used **inverted indexing** to match the query and gather a list of relevant document candidates to calculate similarity. However, this method does not take the semantics of the word into account and expects the word to match directly.

● Inverted indexing can produce extensive indexes that require significant storage space and processing power, especially for collections with many unique terms and documents. This can limit the scalability and efficiency of the indexing and retrieval system and may require additional hardware or infrastructure to handle.

● Index maintenance: Inverted indexing requires frequent updates and maintenance to reflect changes in the collection, such as new documents or modified content. This can be time-consuming and resource-intensive, especially for dynamic or rapidly changing collections, and may require specialized techniques and algorithms to handle efficiently

2. Problem Definition

The vector space model has limitations due to its assumption of orthogonality between words, neglecting their semantic relationships. Our hypothesis is that by modeling relationships between words and considering their context, we can enhance the performance of our search engine. Our project aims to explore different models to address the limitations of the vector space model, including **sequential modeling** to generate **embeddings** and estimate relevance/similarity functions using **neural networks**. We also aim to retrieve relevant documents for the query embedding by exploiting the **clustering** of similar documents in high dimensional space. Our evaluation metrics mainly focus on **MAP** and **nDCG**, and our ultimate goal is to improve the search engine on the cranfield dataset.

3. Motivation

During the development of our search engine using a basic vector space model, we discovered several inconsistencies in the search results, highlighting the limitations of relying solely on this model, as discussed in section 1.1. Our task for this project was to construct a prototype search engine using the Carnfield dataset and the Vector Space Model, which, as previously mentioned, has certain constraints. Our objective is to enhance the performance of our search engine by exploring various models and techniques that we acquired in class. We have attempted several strategies, including:

- Latent Semantic Analysis
- Word Embeddings (Word2Vec, FastText)
- Generating sequential embeddings with RNNs
- Generating sequential embeddings with a pretrained BERT model
- Using clustering to retrieve documents

3.1 Motivation for Latent Semantics Analysis

Latent Semantic Analysis (LSA) is a widely used method for generating word embeddings that capture the contextual similarity between words. The motivation behind using LSA for generating word embeddings lies in its ability to exploit the statistical relationships between words in a large corpus of text. By constructing a Pointwise Mutual Information (PMI) matrix between terms and performing Singular Value Decomposition (SVD), LSA reduces the dimensionality of the term space and maps the words onto a lower-dimensional space. This process captures the latent semantic relationships between words, allowing for the generation of word embeddings that are effective at capturing the contextual similarity between words.

3.2 Motivation for Word Embeddings (Word2Vec, FastText)

Word embeddings generated by methods like Word2Vec and FastText have shown to be effective in capturing semantic relationships between words, making them a popular choice for Information Retrieval Systems (IRS). These methods utilize neural networks to learn the word representations from a large corpus of text, and the resulting embeddings can capture the contextual similarity between words. The motivation behind using these methods for IRS lies in their ability to generate meaningful representations of text that can be used for a variety of NLP tasks. By representing words as dense vectors in a high-dimensional space, Word2Vec and FastText can capture semantic relationships between words and are effective at modeling complex language structures. These methods have shown to be particularly effective at capturing word relationships that may be missed by traditional keyword-based methods, making them a valuable tool for improving the performance of IRS.

3.3 *Motivation for Generating Sequential Embeddings with RNNs*

In information retrieval systems (IRS), the ability to understand the context and meaning of text data is crucial for improving the accuracy and relevance of search results. Generating sequential embeddings using recurrent neural networks (RNNs) is one approach that has gained popularity in recent years for addressing this issue. RNNs are capable of processing sequential data, such as text, in a way that captures the temporal dependencies between words and phrases. This enables them to generate contextualized embeddings that represent the meaning of a word or phrase based on its surrounding context. By incorporating these embeddings into an IRS, search queries can be matched more accurately with relevant documents, resulting in more precise search results. Additionally, RNNs can be trained on large amounts of data, allowing them to learn complex relationships between words and generate high-quality embeddings that are effective for a variety of applications in IRS.

3.4 Motivation for Generating Sequential Embeddings with a Pretrained BERT Model

In recent years, pretrained language models such as BERT have gained significant attention in the field of natural language processing. These models are trained on large amounts of text data and can generate high-quality contextualized word embeddings. Using a pretrained BERT model to generate sequential embeddings can be especially useful for information retrieval systems as it allows for capturing the context of a word within a sentence or document. By considering the surrounding words in the input sequence, BERT is able to generate embeddings that capture both the syntactic and semantic relationships between words. Additionally, BERT-based models have achieved state-of-the-art performance on a wide range of natural language processing tasks, making them a promising option for improving the effectiveness of information retrieval systems.

3.5 Motivation for Using Clustering to retrieve documents

Inverted indexing is a popular technique used in information retrieval systems to efficiently retrieve relevant documents based on queries. However, one of the limitations of inverted indexing is that it requires pre-computation and storage of a large index, which can be a computationally expensive and time-consuming process. Additionally, inverted indexing may not perform well when dealing with high-dimensional embeddings that result from newer techniques like deep learning models.

In order to address these limitations, clustering has been proposed as an alternative approach for document retrieval in information retrieval systems. Clustering involves grouping similar documents together based on their embedding representations in a high-dimensional space. By using clustering, retrieval can be performed by simply identifying the cluster(s) that contain the most similar documents to the query, thus avoiding the need for a pre-built index. This approach is particularly useful when working with high-dimensional embeddings, as clustering can be more efficient and scalable than inverted indexing.

4. Proposed Methodology

In order to assess the quality of embeddings in our context, we constructed a neural network that takes in input X, where X is a concatenation of document embedding vector and query embedding vector, and y, where y represents the relevance score and can take on values 1, 2, 3, or 4. We trained this network and evaluated its performance using validation accuracies. Based on these results, we selected the embeddings that yielded the highest accuracy for further analysis. It is important to note that we used pretrained embeddings as our dataset was small and training our own embeddings would not have yielded significant improvements in performance.

Given below is the model definition (implemented in tensorflow)

```
doc_shape = doc_embeddings.shape[1:]

query_shape = query_embeddings.shape[1:]

doc_inputs = tf.keras.layers.Input(shape=doc_shape)

query_inputs = tf.keras.layers.Input(shape=query_shape)



doc = tf.keras.layers.Dense(128, activation='relu')(doc_inputs)

doc = tf.keras.layers.Dense(128, activation='relu')(doc)

query = tf.keras.layers.Dense(128, activation='relu')(query_inputs)

query = tf.keras.layers.Dense(128, activation='relu')(query)



x = tf.keras.layers.Concatenate()([doc, query])

x = tf.keras.layers.Dense(128, activation='relu')(x)

x = tf.keras.layers.Dense(128, activation='relu')(x)



outputs = tf.keras.layers.Dense(4, activation='softmax')(x)



opt = Adam(learning_rate=1e-6)
```

```
model = tf.keras.Model(inputs=[doc_inputs, query_inputs], outputs=outputs)

model.compile(loss='categorical_crossentropy', optimizer=opt,
metrics=['accuracy'])
```

4.1 Latent Semantic Analysis

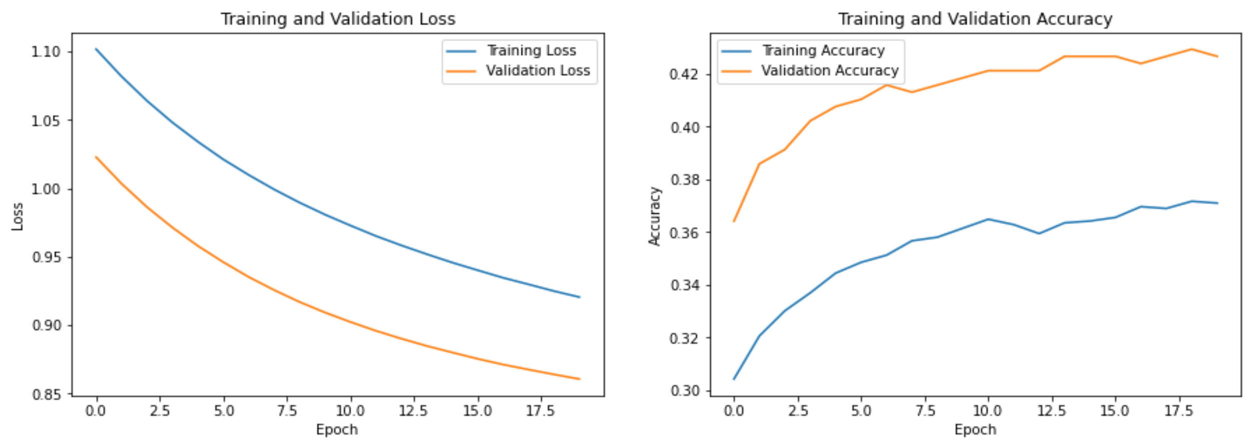
Latent Semantic Analysis (LSA) is a technique used to generate word embeddings by capturing the contextual similarity between words. To perform LSA, we start by constructing a term-document matrix that represents the frequency of occurrence of each term in each document of a large corpus of text. From this matrix, we can compute a pointwise mutual information (PMI) matrix that captures the statistical relationships between terms based on their co-occurrence in the documents.

Next, we perform singular value decomposition (SVD) on the PMI matrix, which factorizes the matrix into three matrices: U, S, and V. The U matrix contains the left singular vectors that represent the document vectors, the V matrix contains the right singular vectors that represent the term vectors, and the S matrix contains the singular values that represent the strength of the relationship between the document and term vectors.

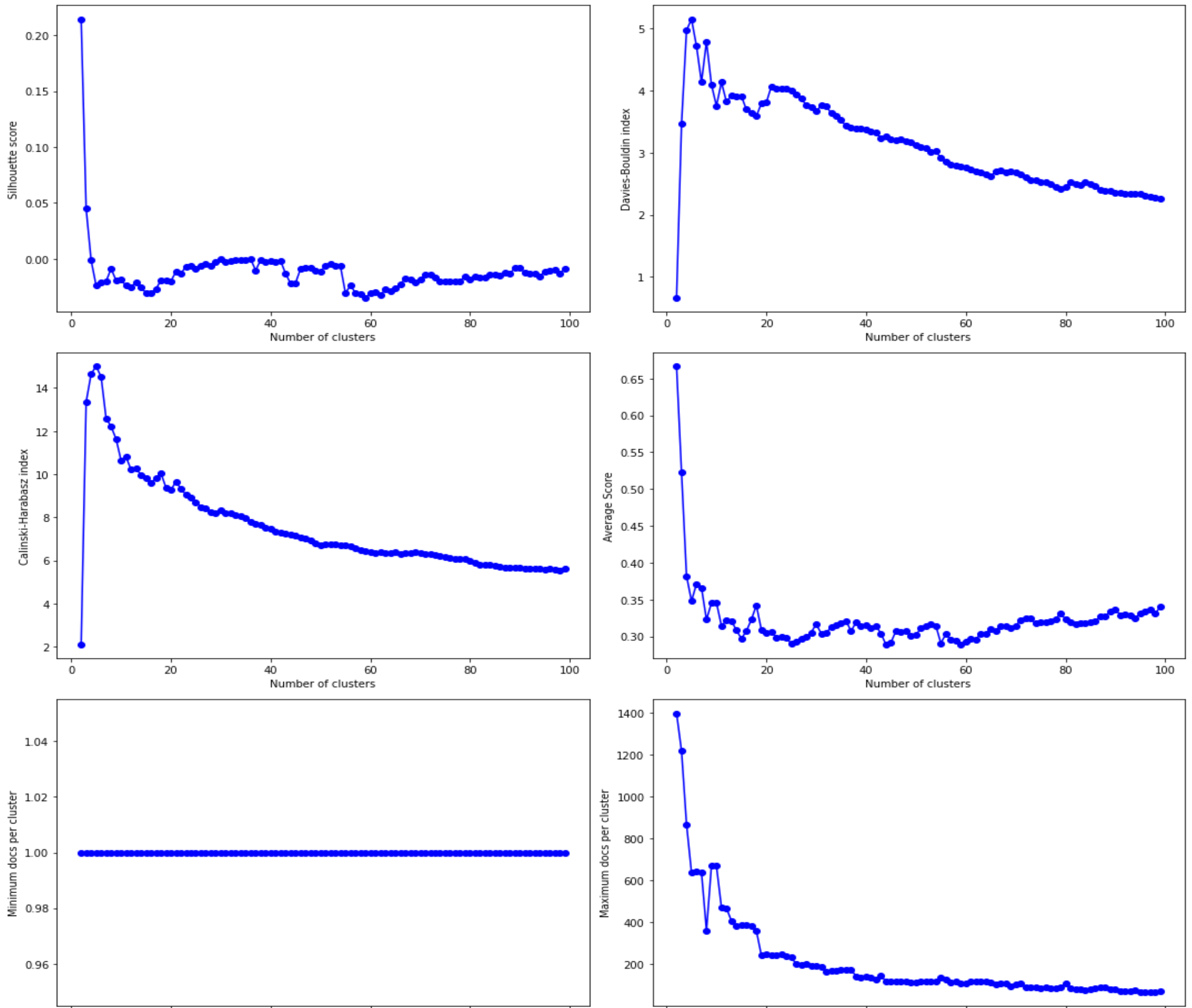
By truncating the SVD matrices to a smaller rank, we can reduce the dimensionality of the term space and obtain a lower-dimensional representation of the original matrix. This representation captures the latent semantic relationships between terms and documents, allowing for the generation of word embeddings that can effectively capture the contextual similarity between words.

To prepare a dictionary that maps the centroid of clusters of documents to the document IDs belonging to that cluster, we tested the embeddings for cluster index. To cluster the document embeddings, we chose to use the Expectation-Maximization (EM) algorithm, which is commonly used for unsupervised clustering tasks. EM is a widely-used algorithm that assumes that data is generated by a probabilistic model with hidden variables. The algorithm estimates the parameters of the model using a combination of a maximization step, which estimates the parameters of the model given the observed data and the current values of the parameters, and an expectation step, which computes the expected value of the hidden variables given the current estimate of the parameters.

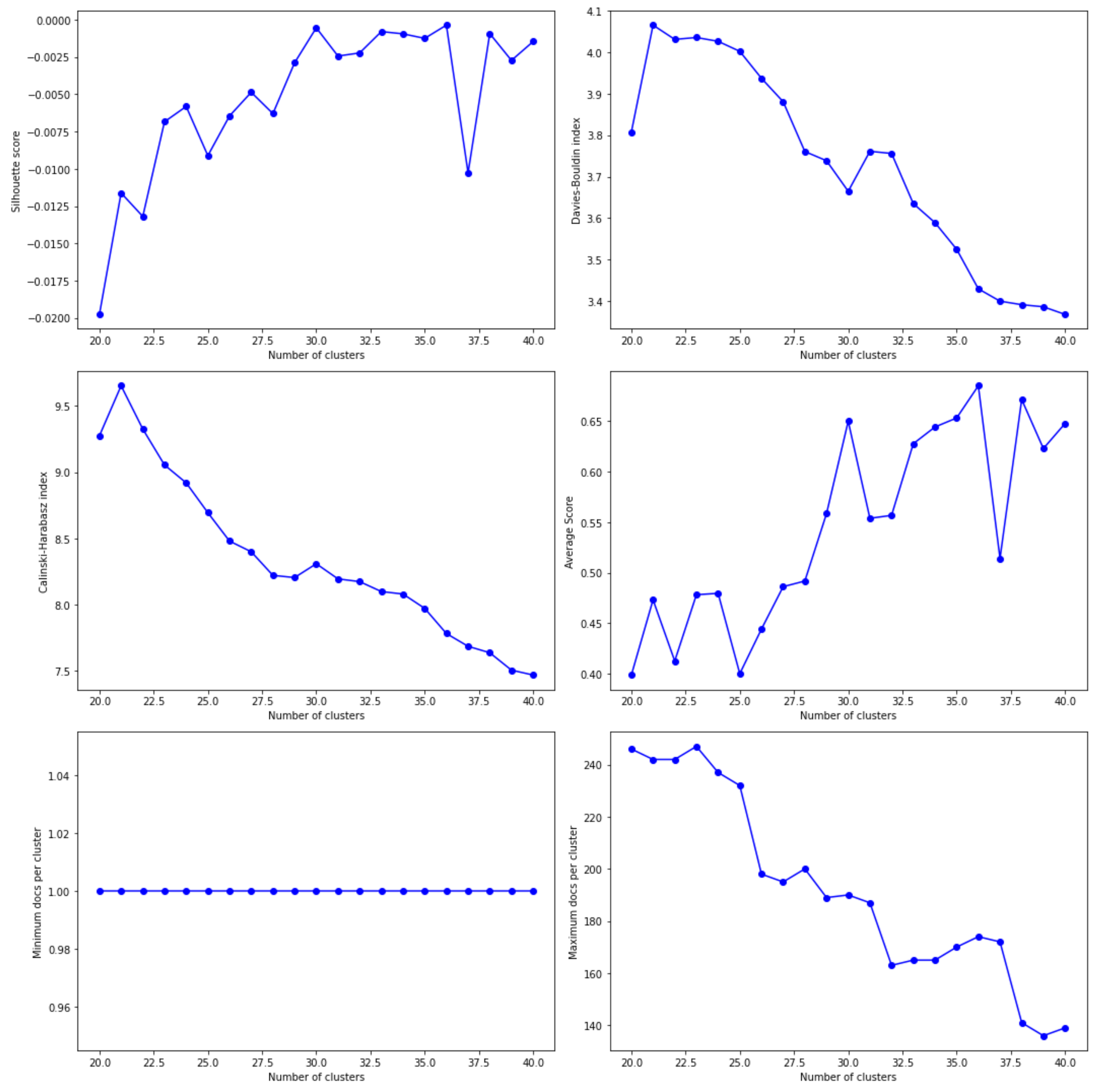
Extrinsic Evaluation results were as follows:



To evaluate the clustering performance, we used an average of three different metrics: silhouette, Calinski-Harabasz, and Davies-Bouldin. Silhouette score measures the similarity of an object to its own cluster compared to other clusters, with a range of [-1, 1] and higher values indicating better clustering results. The Calinski-Harabasz score computes the ratio of between-cluster variance to the within-cluster variance, with higher scores indicating better separation between clusters. The Davies-Bouldin score measures the average similarity between each cluster and its most similar cluster, with lower scores indicating better clustering results.

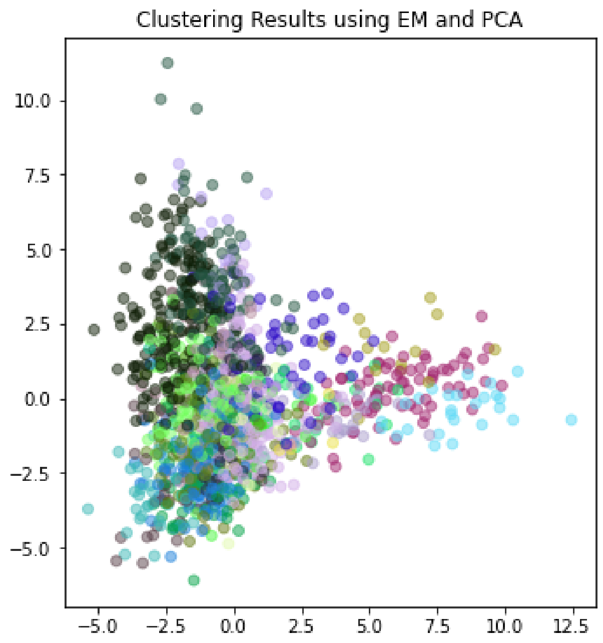


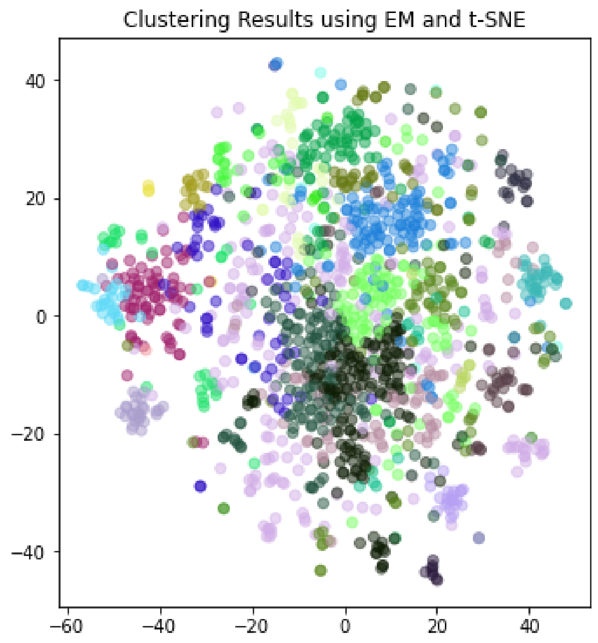
We plotted the average score against the number of clusters to determine the optimal number of clusters for our document set. As we have noticed intriguing and significant outcomes within the range of 20 to 42, we have conducted the experiment and provided an enlarged plot for better visualization.



We determine that the optimal number of clusters is 30 based on both the average score and the maximum number of documents falling within this range.

After applying the EM algorithm for clustering and labeling the data, we used PCA and t-SNE for generating a 2D representation of the embeddings. The results obtained for 30 clusters are presented below.





Principal component analysis (PCA) and t-distributed stochastic neighbor embedding (t-SNE) are commonly used for visualizing high-dimensional embeddings in 2D space. This allows for a more intuitive understanding of the relationships between documents based on their embeddings. PCA is a linear dimensionality reduction technique that aims to capture the most important variation in the data while minimizing the loss of information. On the other hand, t-SNE is a nonlinear technique that aims to preserve the pairwise similarities between data points in the low-dimensional space. Both techniques can help in identifying clusters of similar documents and can aid in the interpretation and evaluation of document embeddings for information retrieval tasks.

Since PCA is not good at capturing non linear relationships, When using t-SNE, one can observe clusters of documents in the 2D space, which may correspond to topics or themes. The distance between two points in the 2D space is a reflection of the similarity between the corresponding documents in the high-dimensional space. Hence, the closer the documents are in the 2D space, the more similar they are in terms of their embeddings. This information can be used to understand the relationships between different documents and to identify groups of documents that are similar to each other.

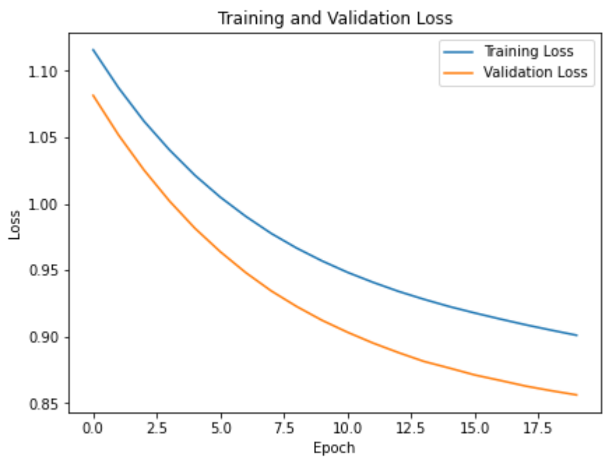
4.2 Word Embeddings (Word2Vec, FastText)

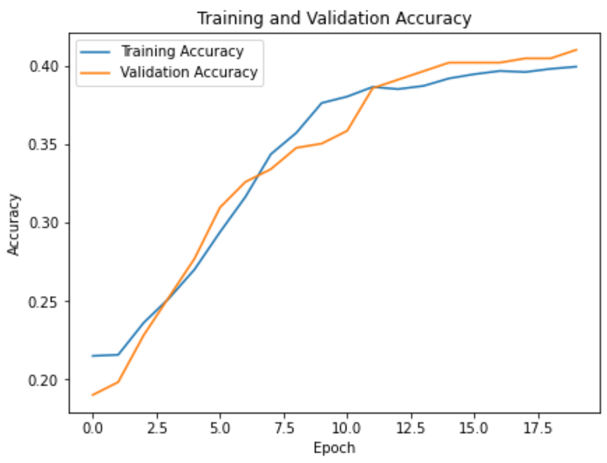
Word2Vec and FastText are two widely used methods for generating word embeddings. Word2Vec is a neural network-based model that takes a large corpus of text as input and learns the vector representations of words by predicting the context in which each word appears. The model has two training methods, Continuous Bag of Words (CBOW) and Skip-gram. In the CBOW method, the model predicts the target word based on its surrounding context words, while in the Skip-gram method, the model predicts the context words based on a target word. Once the model is trained, the learned weights for each word form the word embeddings.

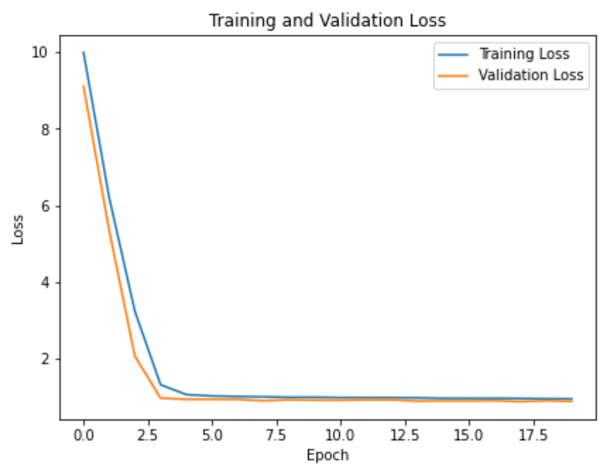
FastText is a variant of Word2Vec that not only learns embeddings for individual words but also for character n-grams. This allows the model to generate embeddings for words that are not present in the training data but are composed of similar n-grams as other words in the dataset. FastText also uses the CBOW and Skip-gram methods for training.

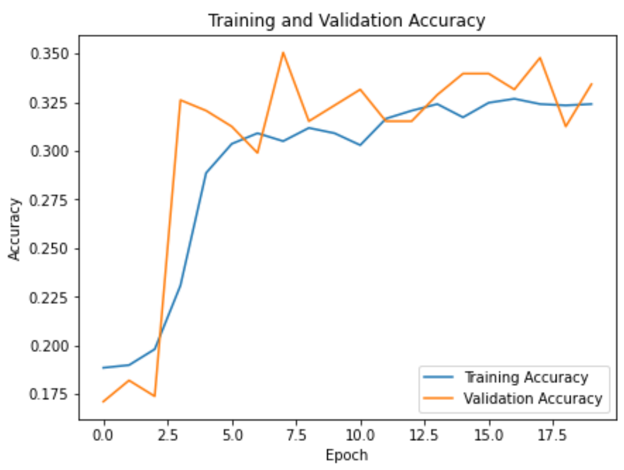
In our study, we did not train our own Word2Vec or FastText models since the dataset was relatively small. Instead, we used pre-trained vectors provided by the respective models.

Firstly, we evaluate Word2Vec and FastText using an extrinsic evaluation approach.





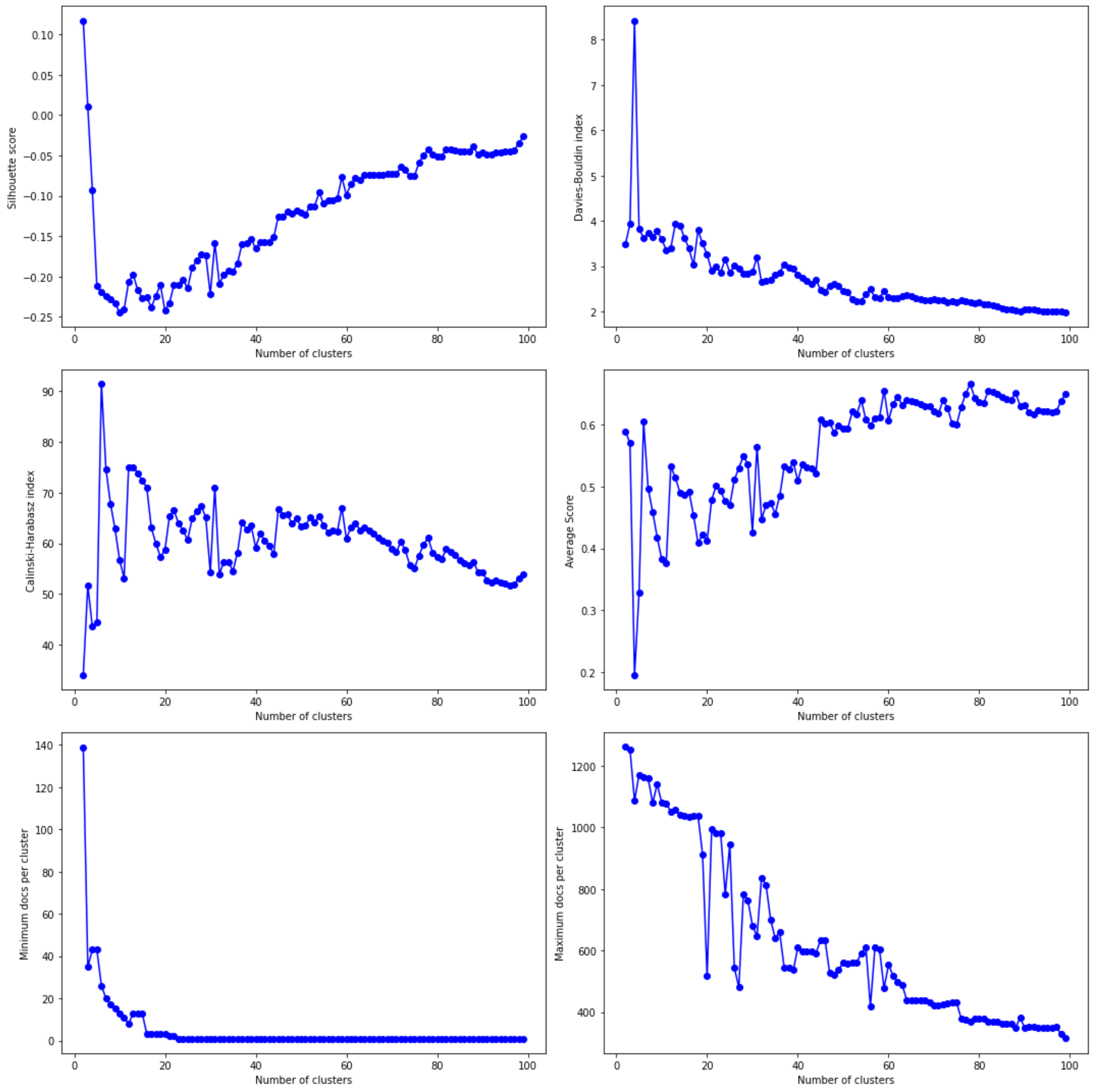




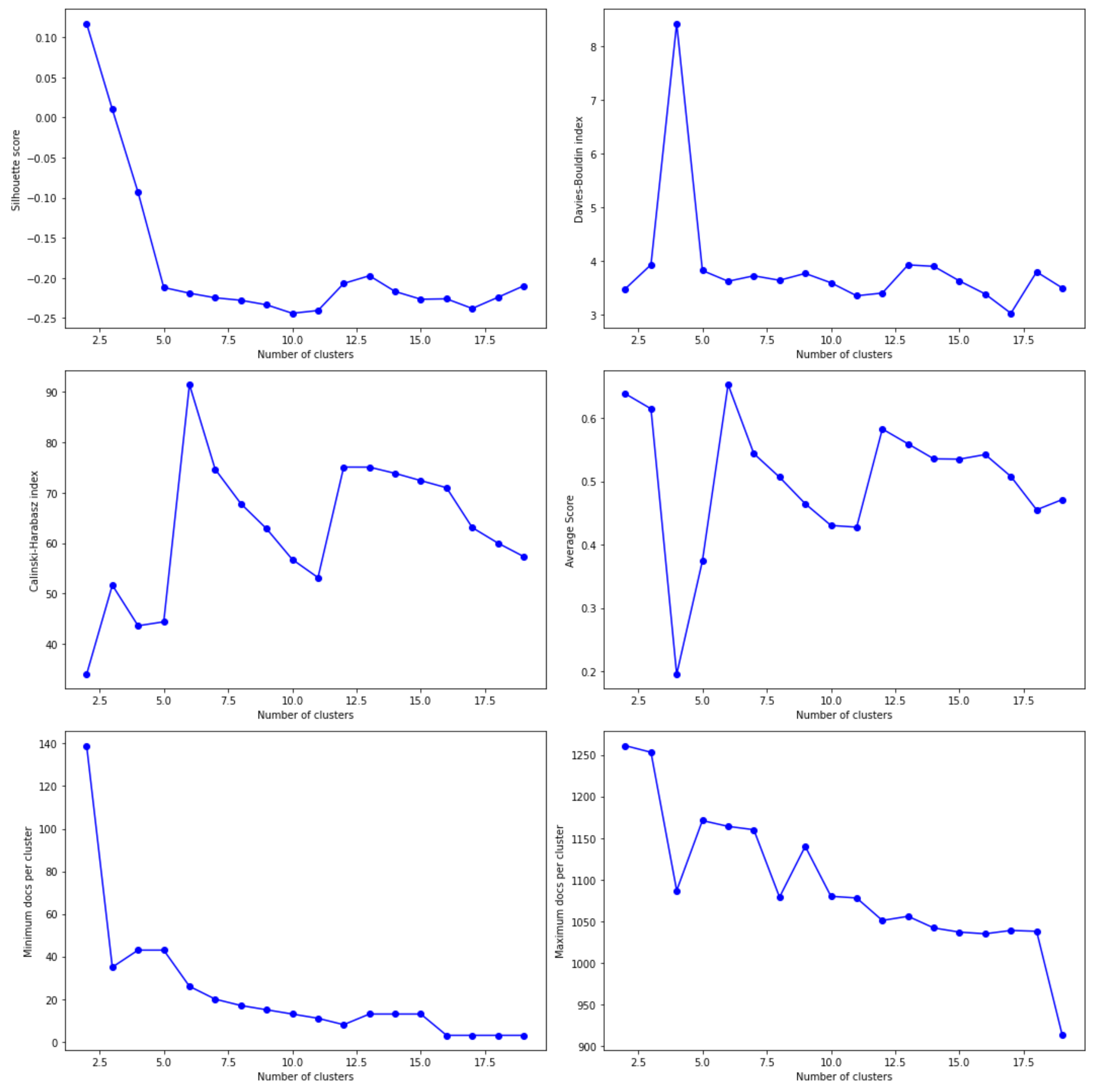
(Plotted results for word2vec and fasttext respectively)

Due to the unsatisfactory performance of fastText on the classification task, we select word2vec as our preferred embedding method.

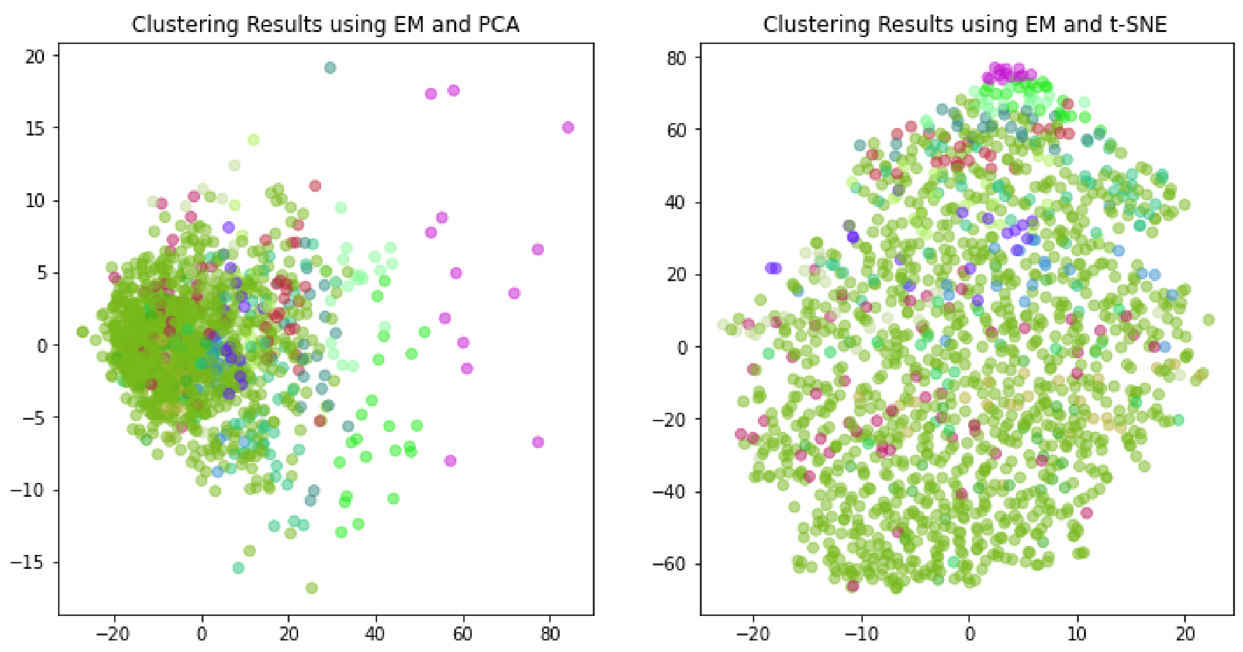
We have repeated similar experiments and plotted the graphs that were presented earlier.



As we have noticed intriguing and significant outcomes within the range of 2 to 20, we have conducted the experiment and provided an enlarged plot for better visualization.



We determine that the optimal number of clusters is 15 based on both the average score and the maximum number of documents falling within this range.



Both plots indicate the presence of a single large cluster with over 1,000 documents.

4.3 Generating sequential embeddings with RNNs

Generating document embeddings using Recurrent Neural Networks (RNNs) involves treating a document as a sequence of tokens (words). The RNNs learn to capture the dependencies between the words in the sequence and produce a fixed-length vector representation of the entire document. One popular type of RNN for generating document embeddings is the Long Short-Term Memory (LSTM) network.

The LSTM network has a memory cell that can selectively forget or remember information over time. This ability to selectively retain relevant information and forget irrelevant information makes it suitable for generating document embeddings that capture the context of a sequence of words.

To generate document embeddings using RNNs, the tokens in a document are first converted into vectors using an embedding layer. The RNN then processes the sequence of embeddings, updating its memory cell at each time step. Finally, the output of the last time step is used as the fixed-length document embedding. This process can be trained on a large corpus of documents using techniques such as backpropagation through time (BPTT) to learn the parameters of the RNN and the embedding layer.

Document embeddings generated using RNNs have shown promising results in tasks such as document classification, clustering, and information retrieval. These embeddings can capture the context of a document more effectively than traditional methods such as bag-of-words or TF-IDF, where word order and context are ignored.

To generate document embeddings, we trained a model on a classification task and extracted the hidden states of LSTMs as document and query embeddings using the trained weights and forward propagation. However, the (q_id, doc_id, q_rel) tuples were not sufficient for our purposes, so we created a new dataset with 225 queries and 1,400 documents. For every pair in the q_rel file, we labeled it as 1 for relevant and 0 for not relevant, and trained this as a regression problem to denote the output as the similarity between the document and query sequences.

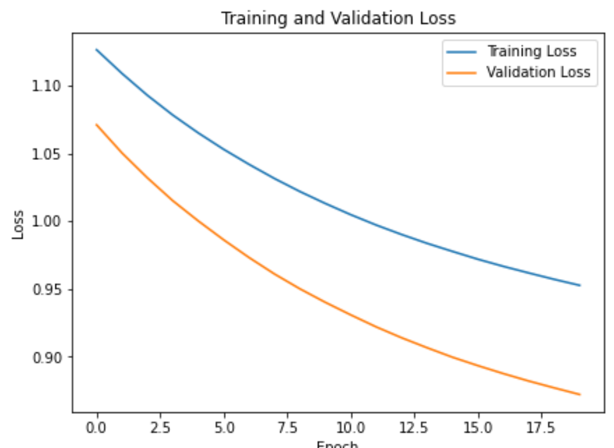
Following is an implementation of model in tensorflow

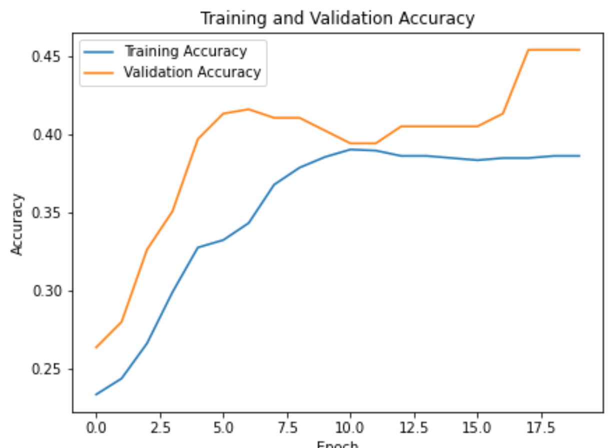
```
HIDDEN_SIZE = 128
VOCAB_SIZE = 9000
EMBEDDING_DIM = 64
DENSE_SIZE = 32

doc_input = tf.keras.layers.Input(shape=(None,), dtype="int32",
name="doc_input")
query_input = tf.keras.layers.Input(shape=(None,), dtype="int32",
name="query_input")

embedding_layer = tf.keras.layers.Embedding(input_dim=VOCAB_SIZE,
output_dim=EMBEDDING_DIM)

doc_embedding = embedding_layer(doc_input)
doc_dense_layer = tf.keras.layers.Dense(DENSE_SIZE,
activation="relu")(doc_embedding)
doc_lstm_layer = tf.keras.layers.LSTM(HIDDEN_SIZE,
return_state=True)(doc_dense_layer)
doc_encoder, doc_hidden_state, doc_cell_state = doc_lstm_layer

query_embedding = embedding_layer(query_input)
query_dense_layer = tf.keras.layers.Dense(DENSE_SIZE,
activation="relu")(query_embedding)
query_lstm_layer = tf.keras.layers.LSTM(HIDDEN_SIZE,
return_state=True)(query_dense_layer)
query_encoder, query_hidden_state, query_cell_state = query_lstm_layer
```

```
concatenated = tf.keras.layers.concatenate([doc_encoder, query_encoder])

dense_layer_1 = tf.keras.layers.Dense(DENSE_SIZE,
activation="relu")(concatenated)
dense_layer_2 = tf.keras.layers.Dense(DENSE_SIZE,
activation="relu")(dense_layer_1)

output_layer = tf.keras.layers.Dense(1, activation="sigmoid")(dense_layer_2)

model = tf.keras.models.Model(inputs=[doc_input, query_input],
outputs=output_layer)

# Create new models to get the hidden states
doc_model = tf.keras.models.Model(inputs=doc_input, outputs=[doc_hidden_state,
doc_cell_state])
query_model = tf.keras.models.Model(inputs=query_input,
outputs=[query_hidden_state, query_cell_state])

opt = tf.keras.optimizers.Adam(learning_rate=1e-4)
model.compile(loss="mse", optimizer=opt, metrics=["mse"])

history = model.fit(
    {"doc_input": x_doc_seq, "query_input": x_query_seq},
    y,
    epochs=5,
    batch_size=32,
    validation_split=0.3
)
```

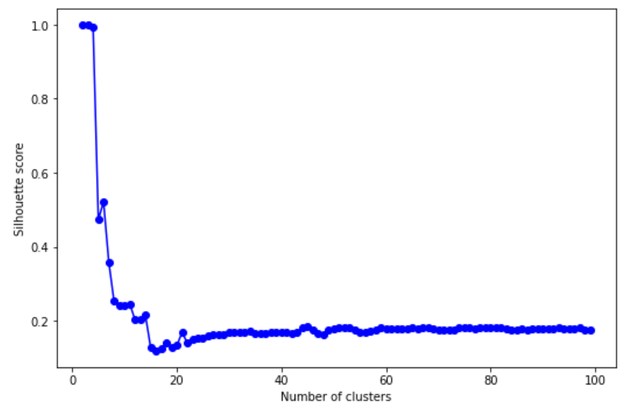
Finally, we use the doc_model and query_model to simply feed forward our sequences and calculate embeddings from the hidden state.

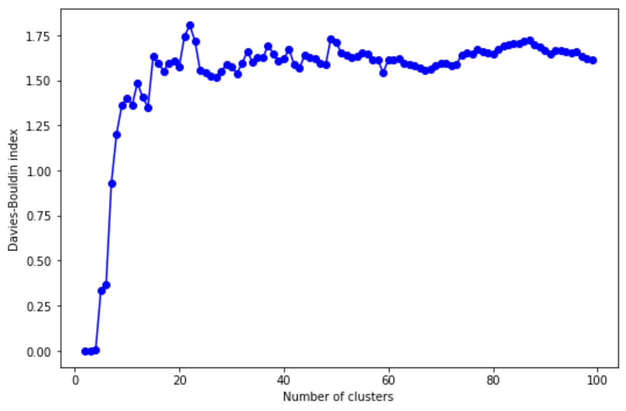
Extrinsic evaluation :

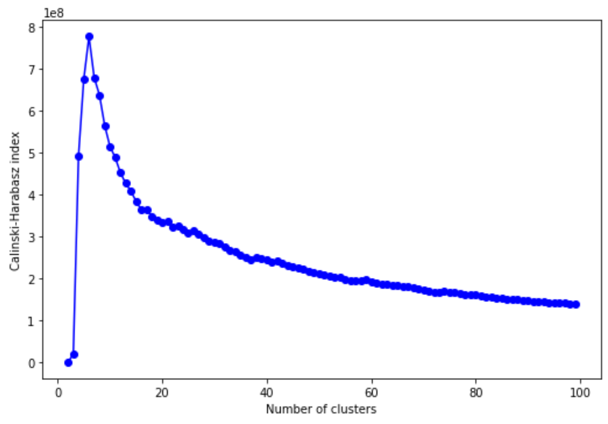


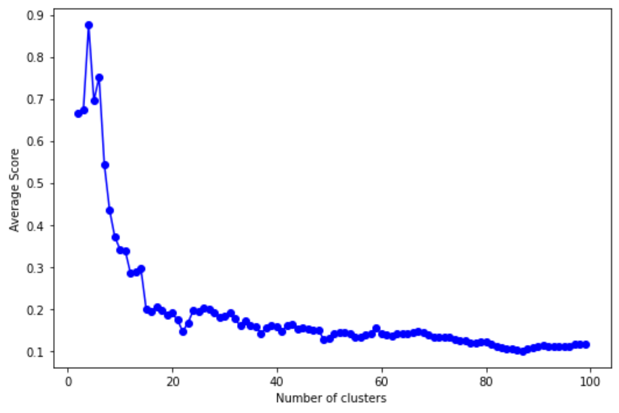


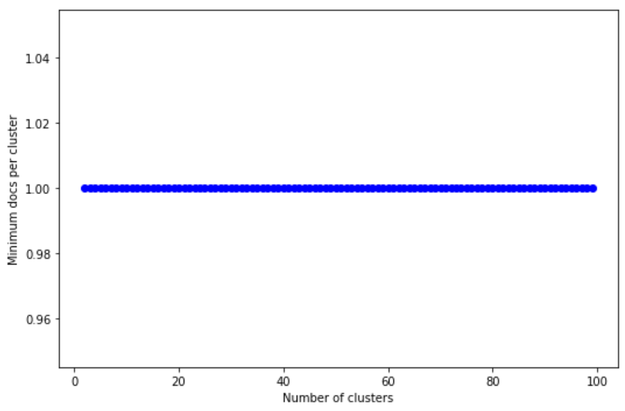
The clustering experiments were repeated and the following results were obtained:

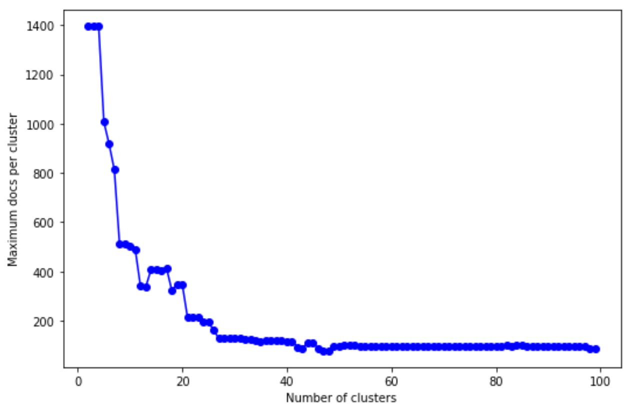




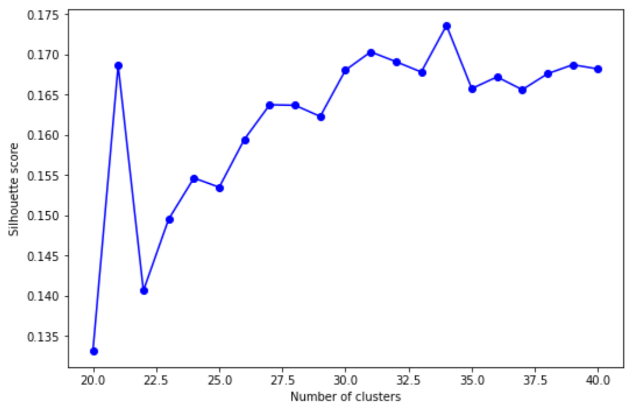


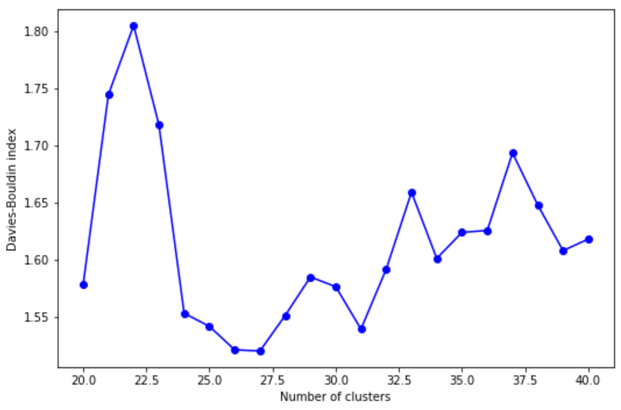


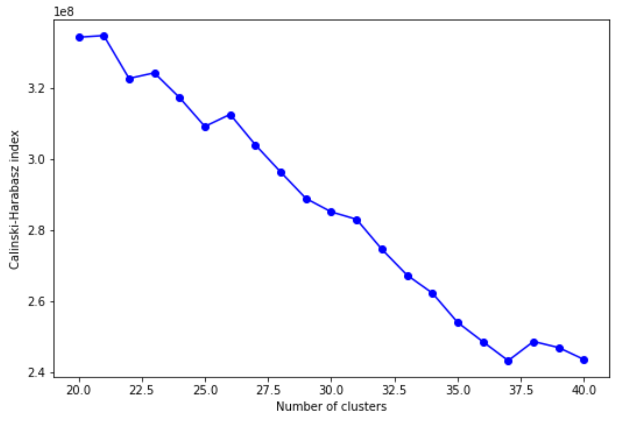


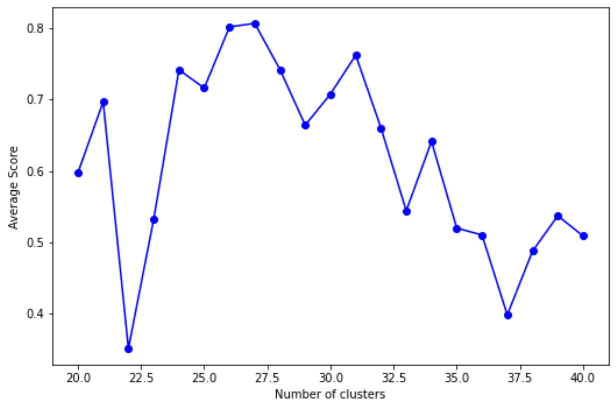


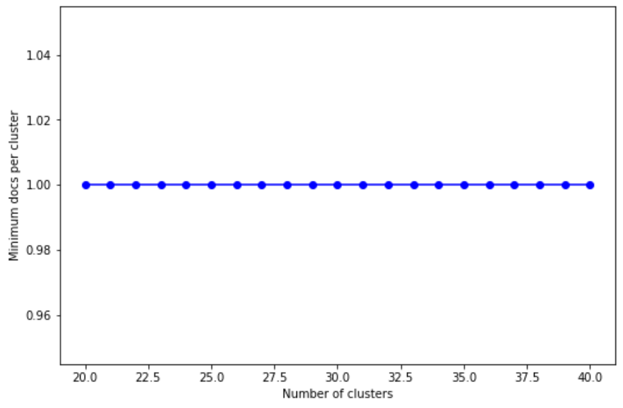
We magnify and plot the results in the 20,40 range.

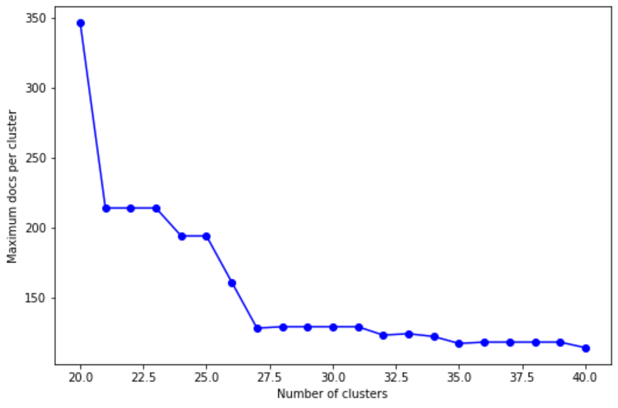




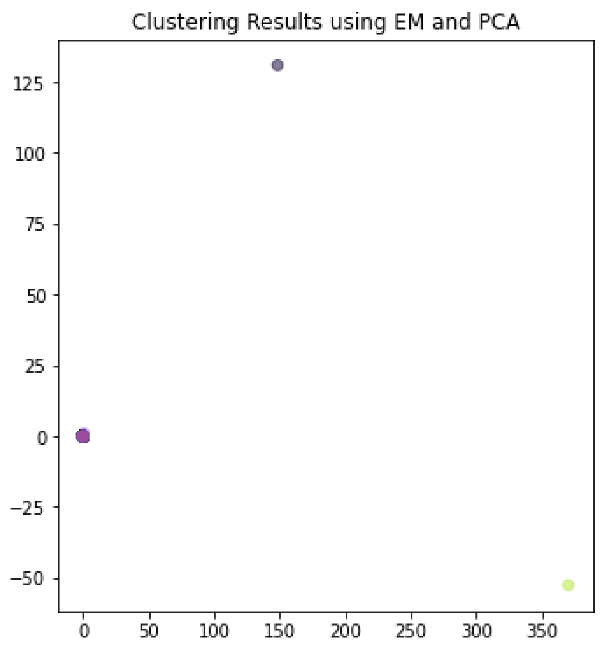


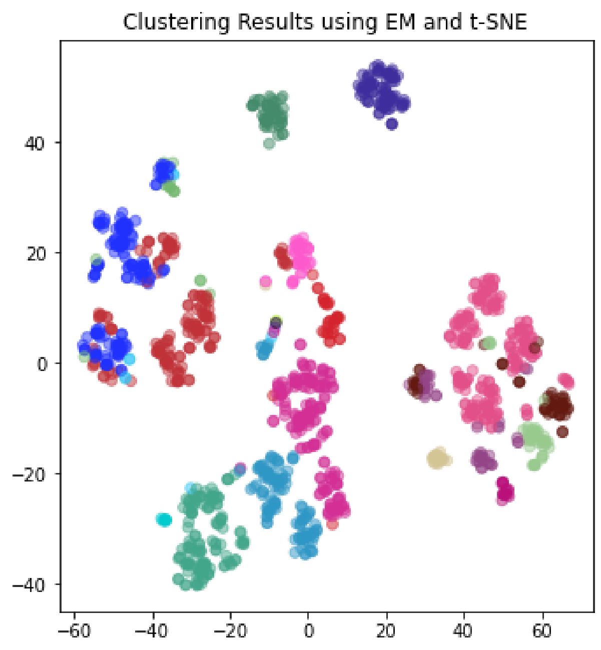






We create a dictionary with k=26 clusters.





The results obtained from reducing the dimensionality of the data using PCA showed some peculiar results, which could be attributed to the method's inefficiency in handling non-linear data. On the other hand, the results obtained from t-SNE showed great clustering performance so far.

4.4 Generating sequential embeddings with Pretrained BERT Model

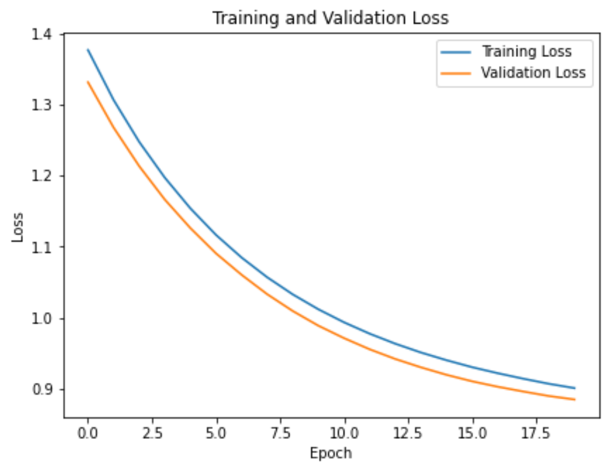
Generating sequential embeddings with a pre-trained BERT model has become a popular technique in information retrieval systems due to its effectiveness in capturing semantic relationships between words and contextual information. BERT stands for Bidirectional Encoder Representations from Transformers, which is a powerful neural network architecture for natural language processing tasks.

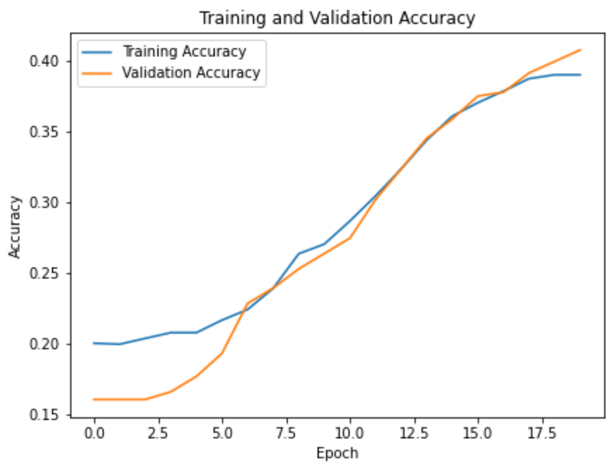
Using a pre-trained BERT model for generating document embeddings involves passing the text through the model and extracting the last hidden state of the model as the document embedding. This approach is beneficial because BERT has been trained on large amounts of text data, making it highly effective in capturing semantic relationships between words and contextual information.

In an information retrieval system, these embeddings can be used to represent both documents and queries. By comparing the embeddings of a query with the embeddings of a document, we can measure the similarity between them and retrieve the most relevant documents.

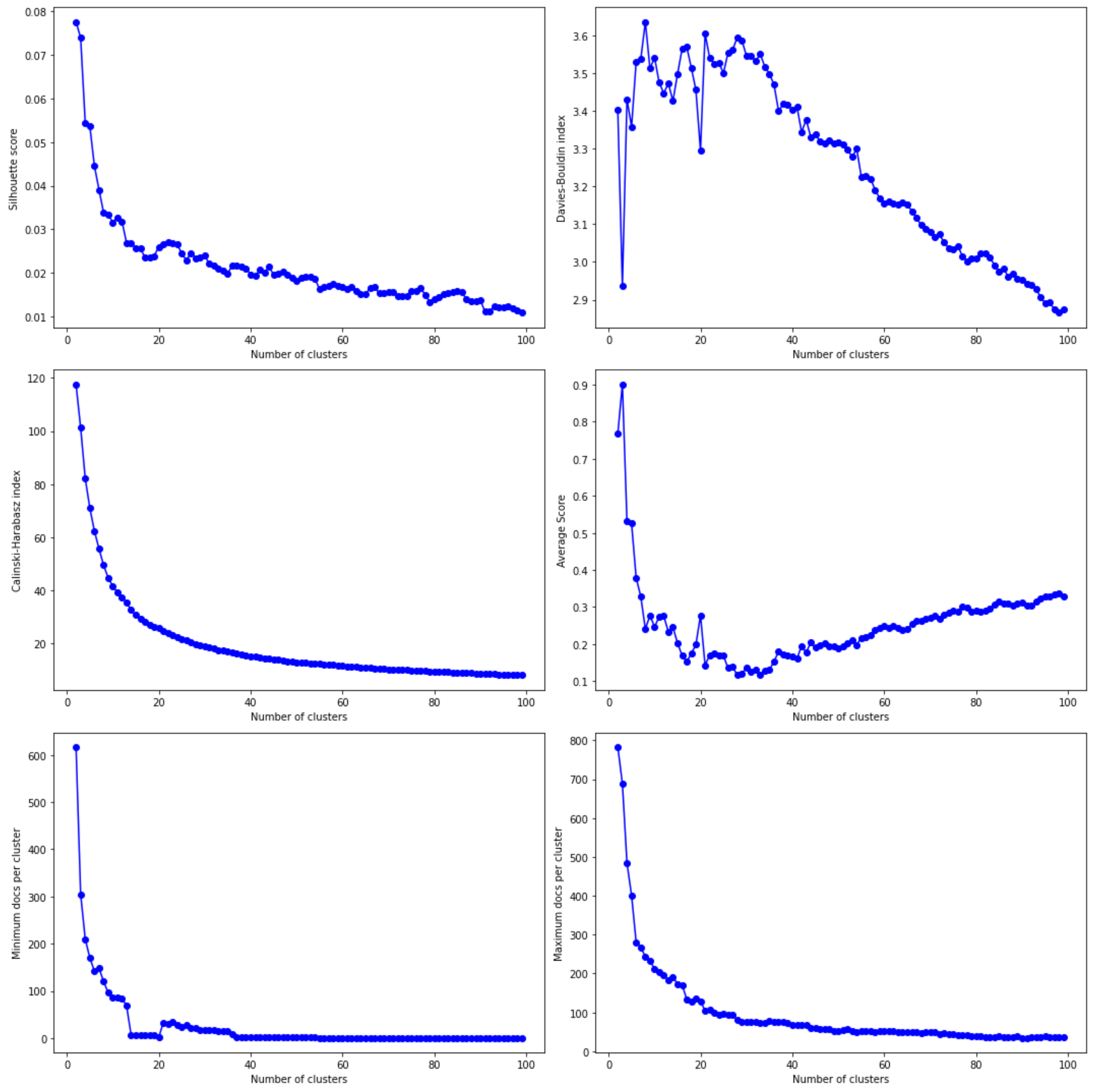
Overall, using a pre-trained BERT model to generate sequential embeddings has shown promising results in information retrieval tasks. The last hidden state of the model is a commonly used method for generating document embeddings, as it has been shown to capture the most relevant information from the input text.

Extrinsic Evaluation :

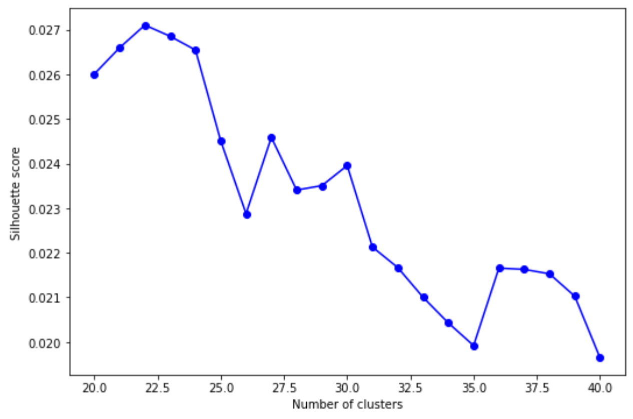


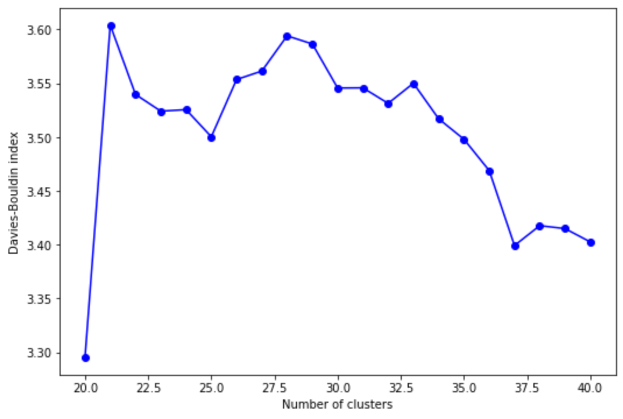


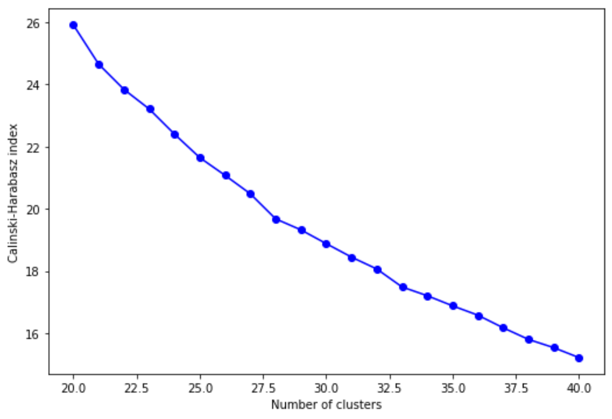
Performing same experiments as above :

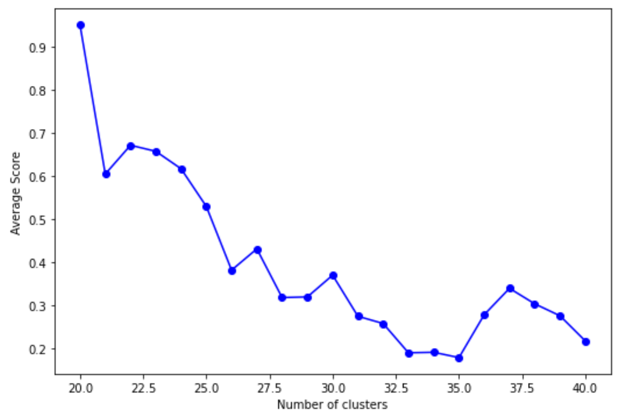


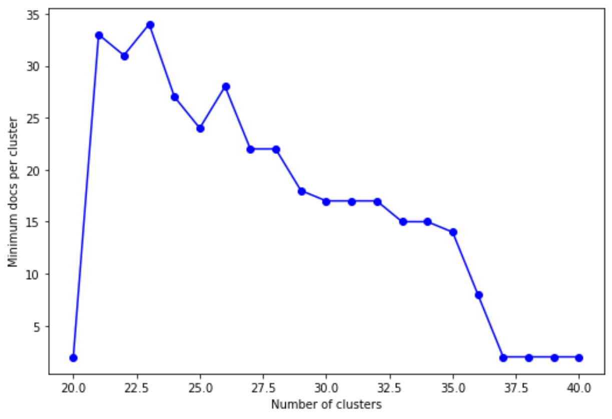
Repeating experiments in 20,41

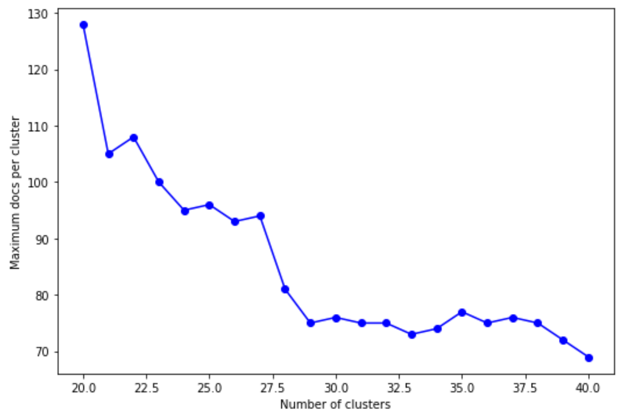




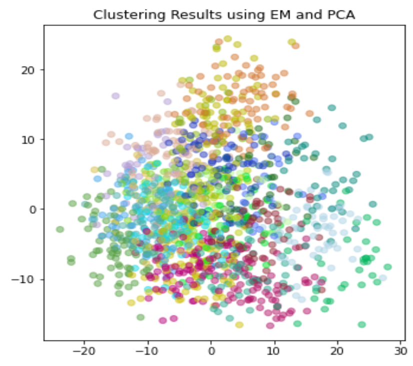


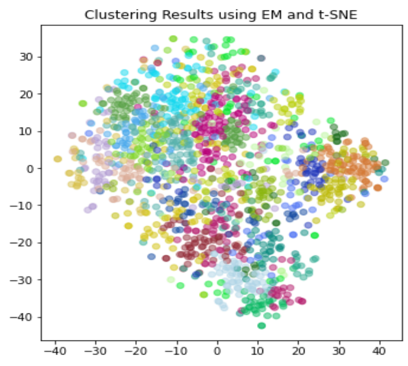






We create a dictionary with k=21 clusters.





5. Evaluation and results

We evaluate the performance of our information retrieval system using two evaluation metrics: Mean Average Precision (MAP) and normalized Discounted Cumulative Gain (nDCG). These metrics are widely used in information retrieval to measure the effectiveness of retrieval systems.

First, we employ inverted indexing with query expansion. This involves using synonyms generated from WordNet to expand the original query and retrieve all relevant documents from the index. By incorporating synonyms, we aim to improve the recall of the retrieval system and ensure that a broader range of relevant documents are retrieved.

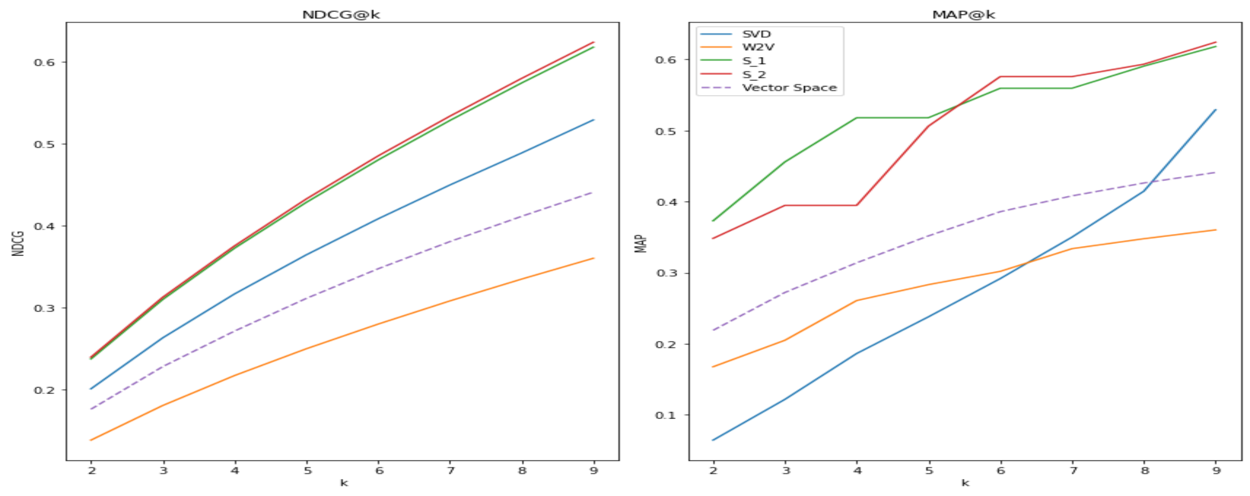
Second, we utilize the centroid index approach. This method involves calculating the centroid representation of the query and retrieving all documents that are similar to the query centroid based on cosine similarity. By clustering documents around query centroids, we aim to retrieve documents that are semantically similar to the query.

For both retrieval approaches, we use the q_rel scores, which represent the relevance judgments of the documents for the given queries. These relevance scores are used to assess the performance of the retrieval system by comparing the retrieved documents with the ground truth judgments.

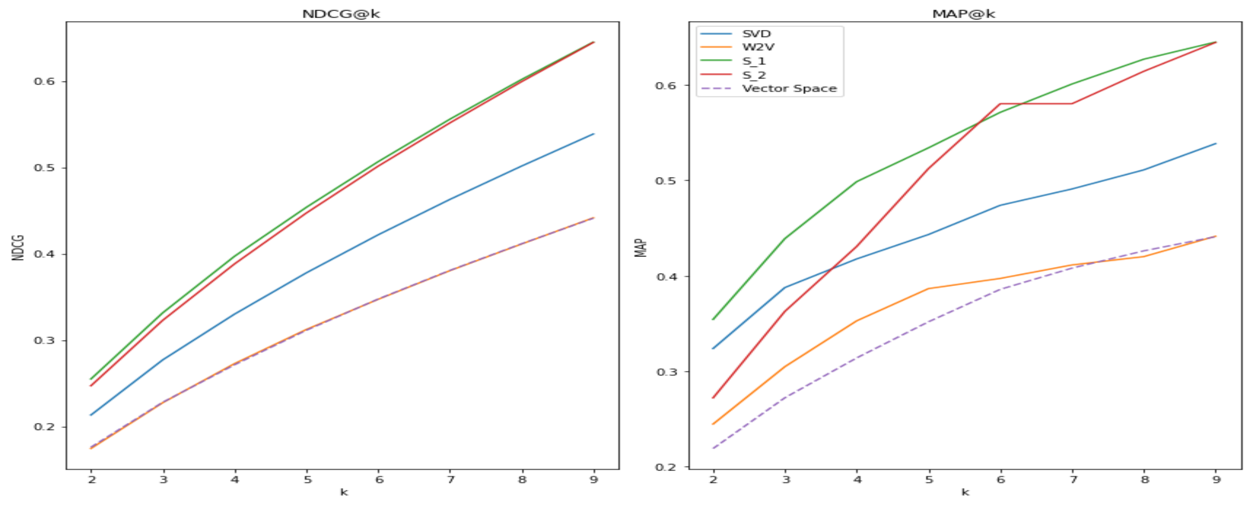
Finally, we calculate the MAP and nDCG scores based on the retrieved documents and their relevance judgments. MAP measures the average precision at different recall levels, taking into account the order of the retrieved documents. nDCG evaluates the quality of the ranking by considering the relevance of the documents and their positions in the ranked list.

By employing these evaluation metrics and retrieval approaches, we can assess the effectiveness and relevance of our information retrieval system in retrieving relevant documents for a given query.

Results with using centroids for retrieving documents



Results with using inverted index for retrieving documents



- The performance of our baseline "Vector Space" model appears to be better than the word2vec model. This observation may be attributed to the use of pretrained vectors, which are not tailored to specific contexts. To address this limitation, we propose a solution by defining a context window around a word and leveraging the vector embeddings of both the context words and the target word. By computing a weighted average of these embeddings, we aim to generate context-specific embeddings that capture the contextual information associated with the word.

- When using centroid-based indexing for document retrieval, it has been observed that sequence models tend to outperform other models. One possible explanation for this is that sequence models, such as recurrent neural networks (RNNs), are effective in capturing the sequential nature of documents and encoding their contextual information. By considering the order and dependencies of words in the documents, sequence models can generate more meaningful embeddings, leading to improved retrieval performance.

- As expected, Latent Semantic Analysis (LSA) performs better than the baseline model. This improvement can be attributed to the creation of a custom Pointwise Mutual Information (PMI) matrix, where we utilized the document as the context window. This approach enabled us to generate more accurate and context-specific embeddings. By incorporating the local document context, LSA captures the semantic relationships between words more effectively, resulting in enhanced performance compared to the baseline model.

We could not estimate a better function for capturing similarities between query and document due to huge class imbalance, which can can be taken care of by :

- Upsampling the minority class: Upsampling involves increasing the representation of the minority class by randomly duplicating instances. However, in this scenario, upsampling the minority class may not be effective since it would still be vastly outnumbered by the majority class.

- Downsampling the majority class: Downsampling involves reducing the number of instances in the majority class to balance the class distribution. However, in our case, downsampling the majority class may lead to a significant loss of data and potentially important information.

- Synthetic data generation: Synthetic data generation techniques, such as SMOTE (Synthetic Minority Over-sampling Technique), can create synthetic instances of the minority class based on the existing data distribution. However, generating synthetic data for the minority class may not be appropriate if it already represents an extremely small proportion of the dataset.

Alternatively, we can address the class imbalance issue by obtaining additional data specifically for the minority class or by utilizing an alternative dataset to train our similarity function.

6. References :

1. https://towardsdatascience.com/document-embedding-techniques-fed3e7a6a25d
2. https://www.analyticsvidhya.com/blog/2022/09/understanding-word-embeddings-and-building-your-first-rnn-model/
3. https://towardsdatascience.com/nlp-extract-contextualized-word-embeddings-from-bert-keras-tf-67ef29f60a7b
4. https://medium.com/analytics-vidhya/combining-word-embeddings-to-form-document-embeddings-9135a66ae0f
5. https://moj-analytical-services.github.io/NLP-guidance/LSA.html
6. https://www.listendata.com/2022/06/pointwise-mutual-information-pmi.html
7. https://kavita-ganesan.com/fasttext-vs-word2vec/
8. https://stats.stackexchange.com/questions/76866/clustering-with-k-means-and-em-how-are-they-related
9. https://medium.com/technology-nineleaps/expectation-maximization-4bb203841757