DEPARTMENT OF COMPUTER SCIENCE

Combining Arguments From Different Sources:

A step Towards Automatic Argument Summarization

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Bachelor of Science in the Faculty of Engineering.

Thursday 4th May, 2023

Abstract

This project aims to develop a system for producing a list of important key points, giving a comprehensive view on a topic by combining arguments gathered from multiple sources. By providing easy access for people to familiarise themselves with both sides of a debate, this attempts to address the political polarization and misinformation which social media not only fuels but exacerbates. This paper investigates methods of computing similarity between sentences using Sentence Transformers, methods of clustering using these similarities, and the identification of the most representative arguments within them. This paper introduces two novel algorithms for clustering both using a similarity matrix, Hierarchical KMeans and Greedy Nearest Neighbour, which out perform many popular methods tested. These algorithms identify on average over 75% of the key points written by an expert debater in the data set.

- Designed a new algorithm, Hierarchical KMeans, which uses a similarity matrix to cluster data points and offers flexibility in how close or separate the clusters returned are.
- Designed another algorithm, Greedy Nearest Neighbour, which adopts a greedy approach to clustering and shows an improvement over Hierarchical KMeans
- Tested both of these algorithms amongst other popular methods and evaluated the findings.
- Researched abstractive and extractive summarization methods and evaluated the the performance of them.
- Developed and tested a final system which given a list of arguments, returns a list of the most critical talking points.

Declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Taught Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, this work is my own work. Work done in collaboration with, or with the assistance of others, is indicated as such. I have identified all material in this dissertation which is not my own work through appropriate referencing and acknowledgement. Where I have quoted or otherwise incorporated material which is the work of others, I have included the source in the references. Any views expressed in the dissertation, other than referenced material, are those of the author.

 Thursday 4th May, 2023

Contents

List of Figures

List of Tables

Ethics Statement

This project did not require ethical review, as determined by my supervisor, Edwin Simpson

Supporting Technologies

- I used the Sentence Transformers library to generate sentence embeddings for arguments https://www.sbert.net/.
- I used the Scikit Learn library to test popular standardized clustering algorithms https://scikit-learn.org.
- The project was developed using Python due to the ease of access to many libraries, like the ones listed above https://www.python.org/
- The UMAP library was used to allow the sentence embeddings to be represented on a graph for one figure https://umap-learn.readthedocs.io/
- The HDBSCAN libarary was used when evaluating the clustering algorithm https://hdbscan.readthedocs.io
- I used the ROUGE python library when evaluating the quality of summaries https://pypi.org/project/rouge-score/

Notation and Acronyms

NLP	:	Natural Language Processing
STS	:	Semantic Textual Similarity
BERT	:	Bidirectional Encoder Representations from Transformers
AAS	:	Automatic Argument Summarization
ROUGE	:	Recall-Oriented Understudy for Gisting Evaluation
HDBSCAN	:	Hierarchical Density-Based Spatial Clustering of Applications with Noise
UMAP	:	Uniform Manifold Approximation and Projection

Chapter 1

Introduction

It is crucial to provide people with balanced arguments for all topics, especially controversial topics, so that they can formulate their own opinion. This is especially important now with the prevalence of misinformation and the rise in political polarization. Gaultney et al. discuss how social media furthers political divides by fueling misinformation which plays to individual biases [5]. Considering this, it is alarming that Head et al. found that 45% of American college students said they find it difficult to determine when an article is fake or real [6]. Understanding the arguments for and against a topic can help address these issues however usually require one to read arguments from numerous sources, including parliamentary debates, scientific papers, news articles or increasingly common: social media. People can often be overwhelmed by the quantity of information available to them when researching a topic on the internet, finding it difficult to effectively identify the main talking points in a timely manner. This is why this project aims to develop a technological approach to automatic argument summarization (AAS), condensing a large list of arguments into the main talking points, by using natural language processing (NLP) models and data analytics to provide the user with a clear, objective and concise overview of them. Reducing the time it takes for one to familiarise themselves with both sides of a debate by providing an alternative to manually researching and collating arguments, instead providing a list of the most important key points identified from a list of arguments gathered from multiple sources. This will offer a great benefit to the general public, especially those wanting to become more knowledgeable and well informed about current affairs and other major issues without dedicating a long time to research them. While this project focuses on summarising arguments about controversial topics, the system developed aims to provide a generalised solution to AAS which could be applied to tackle many other problems.

AAS has many applications, including summarising user reviews, open ended questions in surveys and, as this project aims to tackle, aiding in the research of topics on the internet. In spite of the potential of this topic, it is understudied with little literature tackling the problem of generating an overview of the most important key points made from a list of sentences or arguments. There exist websites which provide provide arguments for both sides of the debate featuring a range of sources for a multitude of topics, for example `www.procon.org`. However, this is an organisation and relies on volunteers to manually research these topics using a wide range of sources, condense the information found and write summaries of these arguments, citing their sources. This is a time consuming task which relies on the volunteers to monitor for new research on each topic, update accordingly, and could potentially be subject to a conscious or unconscious bias. Having said this, software can also be subject to biases, with the way models are trained and the data sets they are trained on greatly influencing the results they produce. This project aims to develop a system for automating this process by clustering arguments based on semantic similarity and summarising each cluster, finding the most representative argument within it.

The main challenges faced in this project are minimising the trade off between conciseness and accuracy when developing a method of clustering arguments together and generating a summary from each cluster which is representative of all the information within it. Arguments may be gathered from multiple sources and it is vital that semantically similar arguments making the same point are grouped together regardless of different phrasing or grammar used. While similar arguments need to be grouped together, all arguments are centered around the same topic and therefore any solution developed needs to be capable of identifying the differences between arguments and the key points they are addressing. The challenge of generating a summary from each cluster is ensuring that all arguments are represented within it and

concisely enough so the user can quickly get appointed with the main talking points. The quality of the summary largely is based on the quality of the clustering and hence development of a good clustering method has been given a higher priority in this project.

The high-level objective of this project is to cluster arguments based on semantic similarity and generate a summary containing all the relevant arguments for a given topic. More specifically, the main aims are:

- Research and survey literature on NLP models which aim to commute the semantic textual similarity (STS) of sentences and paragraphs.
- Test multiple approaches to clustering arguments based on STS and recording my findings
- Using data analysis tools to compare these clustering approaches, evaluating the effectiveness of them.
- Developing a clustering algorithm which improves upon existing methods for this task
- Comparing this novel algorithm against available solutions and quantify the improvement in clustering
- Compare extractive and abstractive text summarization methods, evaluating them to identify the best approach based on common metrics.

Chapter 2

Background

2.1 Sentence Embeddings

Comparing sentences and more specifically calculating the STS between them is a non-trivial problem and there exist multiple approaches to it. Some NLP models transform strings into numerical representations, allowing them to be compared with each other; this is known as the embedding. While this is not necessary, it improves performance, especially when comparing large amounts of data. In this section, I will outline common models which generate these embeddings and the reasons I choose one over the other.

2.1.1 Existing Sentence Embedding models

Tomas Mikolov et al. introduce a method of generating vector representations of words, Word2Vec [11], which utilises a two-layer neural network to learn vocabulary and word associations by training using a large input data set. Once trained, every word corresponds to an unique vector and the relationship between two vectors can be used to calculate the similarity or difference between them. This is a very simple model and can be trained quickly however it has two major flaws. Firstly, some words have multiple meanings depending on their context and this model does not support this as it has one vector representation for each word, no matter the context it is in. Secondly, the model generates word embeddings and while word similarity can be computed, it is not clear how sentences could be compared using this.

Tomas Mikolov and Quoc Le address the latter issue, introducing Doc2Vec [8], which adds another vector to the Word2Vec model. For each sentence the model is trained with, a document vector is generated for it which intends to represent the meaning of the sentence and allow it to be compared with other sentences. While this model performs reasonably well it still suffers from the same issue of not addressing word context within sentences as Word2Vec. This can cause two sentences with the same meaning but paraphrased having their vector representations far away from each other due to the difference in words used and therefore being classified as dissimilar. This is a major issue of this model, especially in the context of this project where arguments from different sources will likely be phrased in slightly different ways, depending on the author of them.

Jacob Devlin et al. address this issue by introducing a language representation model which is able to learn contextual relations between words, Bidirectional Encoder Representations from Transformers (BERT) [4]. It leverages large amounts of training data, over 2.5 billion words from Wikipedia, and trains using a Masked Language Model and Next Sentence Prediction. The Masked Language Model enables bidirectional learning of the model by having the it predict the masked word using the words either side of it. Next Sentence Prediction enables the model to learn about relationships between sentences by predicting whether one sentence follows another or not. This model performs substantially better compared to Doc2Vec as it was trained differently and with a data set over 1000 times the size, and thus can consider the context words are in to better determine whether two sentences are similar or not. However, the only issue with the model is that is does not generate sentence embeddings and therefore calculating similarity between two sentences requires them both to be fed into the network which causes

a large computational overhead and becomes unfeasible when working with a large data set.

Nils Reimers and Iryna Gurevych present Sentence BERT (SBERT) [13], a modification to the BERT network which uses Siamese networks to derive semantically meaningful sentence embeddings. Siamese Networks are neural networks which use the same weights while working together at the same time only with different inputs. The use of these networks vastly increases the speed of computing similarity between sentences by generating embeddings for them instead of requiring each pair of sentences to be fed into the network to compute a similarity score like the BERT model. These embeddings can be used to calculate the STS, for clustering or information retrieval by measuring similarity. SBERT provides a huge performance improvement over BERT enabling the efficient clustering of sentences which will be used in this project.

2.1.2 Comparing Sentence Embeddings

Sentence embeddings generated by the above models are simply high dimension vectors and similarity between arguments is determined by comparing their corresponding embeddings. The first way is by calculating the Euclidean distance between vectors and generating a distance matrix between all arguments in a topic. By comparing these distances, we can see which arguments are most similar to each other with smaller distances indicating more similarity. The main issue with using the Euclidean distance to determine similarity between arguments is that only the magnitude of the distance is considered. The direction of the distance vector is as important if not more because it represents the semantic meaning of the argument. For example, the magnitude of the vector could correlate to the number of words which are in both arguments among other factors while the orientation of the vector could correlate to the context of those words and specifically the meaning they have. Therefore, Euclidean distance does not act as the most reliable distance measure as it best to consider both parts of a vector, especially as this is one of the main benefits of using a SBERT model.

The dot product can be mathematically expressed as the product of the magnitudes of both vectors and the cosine of the angle between them:

$$\mathbf{u} \cdot \mathbf{v} = |\mathbf{u}||\mathbf{v}| \cos(\theta) = \sum_{i=1}^{n} \mathbf{u}_i \mathbf{v}_i \tag{2.1}$$

The size of the angle between two vectors is inversely proportional to the directional similarity of their corresponding arguments and as a result the magnitude of dot product. This offers a great benefit over Euclidean distance because the semantic similarity between the two arguments is now included in calculating the distance between them, making the distance matrix much more accurate and representative. One drawback of the dot product is that it is not normalised and distance vectors with larger magnitudes will have large dot products even if their similarity is lesser. For example, comparing a short argument with a long one may generate an inflated dot score due to the large magnitude of the distance vector between them.

Cosine similarity addresses this issue by only considering the vector direction, independent of the magnitude. It can be seen mathematically as a normalised dot product where only the angle between the two vectors is considered and not magnitude:

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|} = \frac{\sum_{i=1}^{n} \mathbf{u}_i \mathbf{v}_i}{\sqrt{\sum_{i=1}^{n} (\mathbf{u}_i)^2} \sqrt{\sum_{i=1}^{n} (\mathbf{v}_i)^2}} \tag{2.2}$$

This is often better for comparing sentence embeddings because the semantic and contextual similarity between arguments is more important than textual similarity such as if they share the same words or phrasing.

2.2 Related Work

Bar-Haim et al. introduce Key Point Analysis (KPA) [1], a NLP task, where given an input collection of arguments about a certain topic, aims to produce a list of the most prominent key-points. They propose a method of summarising arguments by "mapping them to a short list of talking points, termed

key points". The work in this paper focuses on automatically assigning arguments to a list of given key points, which have been written by an expert debater. They introduce the ArgKP data set which contains 28 topics with 378 key points written in total. With the aim of automatically matching each argument to its corresponding key point, they measured their results by calculating the precision, recall and F1 scores for the key points. The final solution used a BERT-base model which they fine tuned for this specific use case, and compared cosine similarity to compute a match score between arguments and potential key points. The model without fine tuning resulted in a F1 score of 0.4 whereas the fine tuned model scored 0.721. This demonstrates the vast improvements which can be achieved by fine tuning the BERT model to the data set being used in the project. The paper concludes by stating that the task of KPA cannot be solved effectively using unsupervised methods based on standardly available models for generating sentence embeddings and supervised learning methods, for example, fine tuning the model for the specific type of data used, are needed to achieve promising results. While this paper clearly shows that sentence embeddings can be used to compare arguments to key points and rank similarity scores between them, they rely on key points written by an expert debater and require the BERT model to be fine tuned with the data set to deliver adequate results.

This project uses the ArgKP data set, introduced in the paper described above by Bar-Haim et al. [1]. It contains 6,514 arguments for 28 controversial topics with each topic containing between 4 and 11 expert written key points and between 196 and 247 arguments. Each argument is provided with its stance on the topic, positive or negative, and the key point it corresponds to if any. This project will use the key points within a topic to compare to and evaluate how clustering solutions perform. The final aim of the project is to provide a summary, in the form of a list of key points, which closely resembles the ones provided in the data set. One issue with this data set is that some arguments do not correspond to a key point, acting as outliers when compared to the other arguments in their topic. These outliers could pose issues for certain clustering algorithms so they will be omitted from the data set during testing as to not skew any results and ensure evaluation metrics are correct and representative. However, any final solution will be tested with the full data set to ensure that it can deal with these outliers correctly. These outliers could be seen as arguments from illegitimate sources or misinformation which should not be included in the final summary of the topic.

Bar-Haim et al. build upon KPA by developing a method for automatic key point extraction [2]. Their method involves filtering the arguments to only consider single sentences containing below a certain number of tokens and using a model trained on around 30 thousand arguments to compute a quality score to identify "key point candidates". The method from the previous paper [1] is used to match each argument to one key point candidate and candidates with the highest coverage are selected as the final key points. The paper achieved precision scores of 0.752 and 0.792 when matching all arguments to 5 and 10 key points. The paper concluded that the automatic process was able to mimic the analysis of the human expert however found that "manually composed key points tend to be more abstract, and in some cases a single manual point matched several more specific automatically extracted points". This paper demonstrates a method of identifying key points from a list of arguments without clustering and instead filtering and testing potential key points. The results are very good for the ArgKP data set it is trained with however it does not score as highly when testing with other data sets. In addition to this, some key points lack clarity and include unnecessary parts which could be removed to provide a better key point. However, as the key points were extracted from the arguments in the data set, this would not be possible without using an abstractive summarization method. This project differs from this paper as my approach will be to cluster these arguments before summarising each cluster whereas they identified key points from all the arguments and assigned each argument to one of those.

2.3 Existing Clustering Algorithms

The KMeans algorithm clusters data by trying to find cluster centres which reduce the inertia (sum of squares distance within the cluster). For each iteration of the algorithm, all values are assigned to their closest cluster centre and then new cluster centres are created by averaging all values within them. This is repeated until the clusters stabilise, meaning their centres do not change significantly after each iteration, and this can be determined by calculating the difference between the old and new centres and comparing this to a predetermined threshold value. One potential issue which could be encountered when using this algorithm is determining the optimal number of clusters to identify.

Agglomerative clustering avoids this issue by utilising a hierarchical approach to clustering where nested clusters are built by merging or splitting them recursively. Each argument starts in its own cluster and the clusters are merged together based on a provided merging strategy and threshold value. These include, Ward which will merge if the sum of squared differences within all clusters is below the threshold value, similar to the calculation performed in KMeans. Maximum linkage will merge if the largest distance between two clusters is below the threshold value while single linkage will merge if the smallest distance between two clusters is below the threshold value. Average linkage which will only merge if the average distance between all clusters is below the threshold value. It will be interesting to see the effect different merging strategies have on the resulting clusters and which one is able to achieve the best results. While this algorithm addresses the issue KMeans has of finding the optimal number of clusters, it has the issue of determining the optimal threshold value which offers the best performance across all topics in the data set.

Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) [3] is a density based clustering algorithm which tries to group objects together by identifying clusters as areas of high density separated from other clusters by areas of low density in the provided data. This algorithm is better able to deal with outliers and noise compared to KMeans and Agglomerative clustering because it is not obliged to assign every data point to a cluster. KMeans and Agglomerative clustering both assign every data point to a cluster which can lead to them being classified incorrectly, reducing accuracy and skewing other clusters whereas HDBSCAN can omit some data points it deems as outliers. The algorithm has one required parameter, the minimum cluster size, which determines the threshold for a data point to be either classified as an outlier or to create a new cluster.

Community detection is an algorithm provided in the Sentence Transformers module which uses cosine similarity to return a list of clusters based on the parameters provided; these are a threshold value and a minimum community size. This is the first algorithm which uses cosine similarity instead of Euclidean distance to group the arguments together and this means that the threshold value has to be a float between 0 and 1. The algorithm will find communities of arguments where the cosine similarity between them is at least the threshold provided. The higher the threshold, the more similar the arguments have to be to qualify to be in the returned communities. This algorithm will face the same challenge as Agglomerative clustering, determining the optimal threshold value and also minimum community size which offers the best performance across all topics in the data set.

2.4 Existing Evaluation Metrics

Recall-Oriented Understudy for Gisting Evaluation (ROUGE) [9] is a set of metrics which compares the words in the summary against the words in the arguments in the cluster to calculate a score. ROUGE-1 is concerned with counting the overlap of singular words between the summary and the arguments while ROUGE-2 is concerned with counting the overlap of bi-grams between the summary and the arguments. ROUGE-L searches for the longest common sub sequence of words within the summary and the arguments. We can calculate the ROUGE scores of any generated summaries and compare them to the expert written key points in the data set to determine how well the system works. One disadvantage of ROUGE scores is that they may not be very accurate or representative of the quality of abstractive summarization methods because they tend to be phrased differently and may not share many bi-grams or sub sequences of words.

BERTScore is an automatic evaluation metric [17] which computes a similarity score between two input texts by using a BERT model to generate embeddings for them and easily comparing them. I think this method will be more accurate than calculating the ROUGE score because it can utilise the context of the words to determine similarity not merely the presence of them. I predict this will calculate a much more accurate score when comparing abstractive summarization methods compared to the ROUGE score generated for the same summaries.

2.5 Existing Summarization Methods

Abstractive summarization differs from extractive summarization in that it involves generating a new sentence or sentences instead of identifying the most important sections within the input text and returning a subset of it. While both aim to return summaries which encapsulate the essence of all arguments

in the cluster, abstractive summarization often generates shorter summaries and phrases them in a novel way. This is achieved with neural networks trained on huge amounts of data, usually newspaper articles and scientific journals to identify important words, phrases and sequences in text. Jingqing Zhang et al. present a trained model Pegasus [16], which aims to generate accurate and concise abstractive summaries from an input list of sentences. The model was trained on a data set of 1.5 billion articles and 350 million web-pages. The model was trained using an objective called Gap-Sentences Generation which works by removing important sentences from an input document and having the model predict the missing sentence based on the remaining sentences in the document. Training using this objective has made this model capable of generating abstractive summaries from a list of sentences.

Chapter 3

Project Execution

As stated in the supporting technologies section, Python was used to test existing solutions available and develop the final solution with the help of the Scikit Learn module [12] which offers many standardised clustering algorithms and other metrics, and the Sentence Transformers module which provides easy access to SBERT models [4].

3.1 Sentence Embeddings

From the literature available surrounding this topic, it is clear to see that Sentence BERT outperforms all other sentence embedding methods by uniquely being capable to consider the context of words to develop a greater understanding of the meaning of an input sentence. There are numerous pre-trained models available to be used with the Sentence Transformers framework, trained with different data sets offering better performance for certain applications.

3.1.1 Comparing SBERT Models

I conducted an experiment comparing the 5 most popular and highest rated SBERT models to determine the model which delivers the best performance for the data set being used in this project. The clustering of arguments relies on the comparison of sentence embeddings generated by a given model and as such, a better and more accurate model will allow for better results. A metric or score needs to computed representing the performance of each model allowing for the comparison between them. In the data set, each argument corresponds to a key point and this can be seen as the gold standard the final system strives to achieve. For each topic in the data set, sentence embeddings were generated for all arguments and a similarity matrix between them was computed. This was used to compute the average intra-cluster and inter-cluster similarities within each topic using the key point assignment in the data set as reference, and the average results across all topics can be seen in table 3.1. Maximising the intra-cluster similarity whilst simultaneously minimising the inter-cluster similarity will allow for improved accuracy within clusters and better separation between them. Peter Rousseeuw proposes Silhouette Score [15], a metric which calculates a score using both the intra-cluster and inter-cluster similarities or distances to determine how dense and well separated clusters are. While this metric is often used to evaluate clustering methods where correct labels are not known, it was calculated for each model to compare how representative the embeddings it generates are of the arguments. Using a similarity matrix, the Silhouette score is defined mathematically as:

$$Silhouette\ Score = \frac{a - b}{max(a, b)},$$

where a is the average intra-cluster similarity, b is the average inter-cluster similarity (3.1)

All models have high intra-cluster similarity with model "multi-qa-mpnet-base-dot-v1" scoring highest with 0.7213. However, they also have relatively high inter-cluster similarity because there is a large overlap in content between arguments concerning the same topic and hence a significant similarity should be expected. This experiment finds that the "paraphrase-multilingual-MiniLM-L12-v2" model achieves

the highest silhouette score of 0.1642 and therefore performs best for the data set and use case of this project. This model differs from others tested, generating sentence embeddings in a 384 dimensional vector space compared to a 768 dimensional vector space. While this is one distinction between them, I think the major reason this model out performs the rest is because it has been trained with an emphasis on improving its ability to identify paraphrasing in text using data sets including duplicate questions from Quora, StackExchange and Yahoo Answers. This allows it to better identify relations between sets of sentences as is necessary for this project.

Table 3.1: Comparison of the 5 most popular and highest rated SBERT models

Model	Intra-cluster similarity	Inter-cluster similarity	Silhouette Score
all-mpnet-base-v2	0.6847	0.5977	0.1271
multi-qa-mpnet-base-dot-v1	0.7213	0.6558	0.0908
all-distilroberta-v1	0.6577	0.5732	0.1285
all-MiniLM-L12-v2	0.6512	0.5725	0.1209
paraphrase-multilingual-MiniLM-L12-v2	0.6443	0.5385	0.1642

3.1.2 Issues with Sentence Embeddings

As seen in table 3.1, these sentence embeddings face the issue of high inter-cluster similarity. This poses the potential problem of arguments which are phrased similarly but opposing generating similar embeddings. This would lead to algorithms clustering them together wrongly and the final summaries generated missing important key points. An example of this issue is shown below:

ARG1 = "capital punishment remains a good deterrent for serious crime"
ARG2 = "the death penalty is not an effective deterrent to crime"

Cosine similarity computed between ARG1 and ARG2 = 0.7613

ARG3 = "cannabis is a dangerous drug and encourages crime"
ARG4 = "cannabis is a safe medicinal plant, it helps people."

Cosine similarity computed between ARG3 and ARG4 = 0.7237

While these pairs of arguments are completely opposing and contradictory, they are formulated almost identically and share a large percentage of the same words, and the model struggles to identify that they are opposing. Splitting the arguments in the data set into positive and negative, using the stance of each argument, and feeding each set into clustering algorithms independently before merging the results of them would alleviate this issue. However this would require any future uses of this system to provide an accompanying stance with each argument instead of just a list of them as the goal of this project is. An alternative to this would be to utilise Stance Detection models to determine whether an argument is pro, against or neutral towards the topic. However, Myrthe Reuver et al. conclude that this NLP task requires more research and work in order for it to provide sufficiently good results [14].

3.1.3 Fine tuning the SBERT model

Related Papers [1] and [4] both fine tune SBERT models to greatly improve the performance of them and in turn achieve significantly better results on the data sets they test. While the data set is comparatively small to the data the models have been trained on, I conducted an experiment to investigate if fine tuning the highest scoring model, "paraphrase-multilingual-MiniLM-L12-v2", using the data set would increase its performance. Pairs of similar arguments were obtained from the train data set by looping over all topics and arguments within them and identifying unique pairs of arguments sharing the same key point. The model was fed these pairs and trained to improve at identifying features which correlated with argument similarity. The fine tuned model had intra-cluster, inter-cluster and silhouette scores of; 0.8663, 0.7413, and 0.1443 respectively. While there is a significant increase in intra-cluster similarity, the inter-cluster similarity has also increased which negates the benefits of the improvement in intra-cluster similarity. Model performance is measured by the silhouette score and the fine tuned model scores lower than the original model, 0.1443 compared to 0.1642, concluding that fine tuning the model did not

increase performance and in fact decreased it. One potential explanation for this is that only similar arguments were fed into the model during training so while it learned how to identify similar arguments, it did not for dissimilar ones. Considering the results of this experiment, I decided to use the pre-trained model "paraphrase-multilingual-MiniLM-L12-v2" for the extent of this project.

3.1.4 Comparing Sentence Embeddings

I conducted an experiment to compare the difference between using the dot product and cosine similarity as a way of comparing sentence embeddings. I did this by firstly computing a distance matrix between the sentence embeddings generated for all arguments in each topic using the dot product. This was used to then compute the intra-cluster and inter-cluster distances within each topic using the key point assignment in the data set as a reference. Similarly, a similarity matrix was computed between the embeddings using cosine similarity and the intra-cluster and inter-cluster similarities were computed. However, these values cannot be compared with each other directly because one is a distance and the other a similarity between 0 and 1. To compare the difference between the two distance measures, the silhouette score was calculated for each one which provided a normalised value which can be used to compare them. Using cosine similarity resulted in a silhouette score of 0.1642 compared to 0.1609 when using the dot product. This is a relatively small difference and there were far greater differences observed in table 3.1 when comparing different SBERT models. Cosine similarity will be the preferred distance measure to use in this project to try and achieve the best overall performance.

3.2 Analysing Cluster Quality

Every clustering algorithm tested will be evaluated using the same metrics to allow for easy comparison between them. The basis of these metrics are; precision which measures the accuracy of predictions, and recall which measures the completeness of them.

Precision is defined as the ratio of correctly predicted positive classes to all predicted positive classes and can be expressed mathematically as:

$$Precision = \frac{TP}{(TP + FP)},$$

where TP is True Positives, FP is False Positives (3.2)

Recall is defined as the ratio of correctly predicted positives classes to all actually existing positive classes and can be expressed mathematically as:

$$Recall = \frac{TP}{(TP + FN)},$$

where TP is True Positives, FN is False Negatives (3.3)

A truly perfect algorithm would result in full and homogeneous clusters. Fullness would be achieved when recall is 1, meaning all objects of the same class are assigned to a single cluster. When precision is 1, clusters are said to be homogeneous, meaning each cluster only contains objects of the same class. Since this is rarely achieved, we aim to strike a satisfactory balance between these two values. We have to deal with the Precision-Recall Trade-off where a model can be tweaked to increase precision at the cost of a decreasing recall or conversely increasing recall at the cost of a decreasing precision.

The F1 score is the harmonic mean of precision and recall and will be used to measure the performance of each approach. It can be expressed mathematically as:

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3.4}$$

These values are calculated by considering all unique pairings of arguments from all clusters and referencing the data set to determine whether they have been clustered correctly. For example, a true positive

would be where a pair of two arguments in the same cluster share the same key point in the data set and a true negative would be where a pair of arguments are in different clusters and correspond to different key points in the data set. Using this method, the precision, recall and F1 scores can be calculated for any clustering method and its performance compared with other approaches.

3.3 Comparing Clustering Algorithms

The data set contains 28 topics, each with a different number of key points and arguments therefore any approach taken needs to work well for all scenarios. For initial testing purposes, one topic was selected, 'We should subsidize space exploration', to speed up development and be able to try many different approaches. Uniform Manifold Approximation and Projection (UMAP) is a vector dimensionality reduction method [10] which is able to convert the embeddings from a 384 dimension vector space into a 2D space which can be plotted on a graph whilst maintaining the relative global positioning of data points. This is demonstrated in figure 3.1 which shows how the sentence embeddings generated from the arguments in the selected topic are distributed and clustered using the gold standard key point assignment in the data set. There is a general separation into two distinct groups which corresponds to the stance of the arguments with key points 1 through 4 opposing the topic and key points 5 through 9 in agreement with the topic. This seems to suggest that the concern about opposing arguments being clustered together may not be an issue and that the model is able to discern between argument stance. While this may not be an issue, this figure does illustrate the major challenges which will be faced when attempting to cluster this data.

Firstly, while there are two distinct groupings based on stance, there is a lot of overlap between the data points within them. This issue can be easily seen with the data points assigned to key point 5. These data points are spread out across approximately half of the space taken up by the main group and as such the intra-cluster distances of these data points is very high. I think this will cause challenges for all clustering algorithms because either key points will be split into multiple clusters to ensure high precision at the cost of low recall or clusters will contain arguments from multiple key points within them causing high recall but low precision. While precision is prioritised over recall in this project, the final summary generated by the system cannot be too verbose and long and containing duplicate key points.

Secondly, similar to the previous point, the similarity between arguments assigned to the key points in the data set is not consistent between key points. This causes the issue that if high precision is the goal of the clustering, similarity thresholds have to be adjusted to the lowest common denominator. What this means is that if there is a key point in the data set where the similarity between the arguments is very high, the global similarity threshold used with a clustering algorithm will have to be set very high as well to ensure that the key point is identified. While the key point gets identified, the number of duplicate clusters has undoubtedly increased and this begs the question if the trade off is worth it.

Additionally, there is a huge variance in the frequency of arguments assigned to each key point within the data set as seen in figure 3.2. For example, there are 71 arguments assigned to key point 3 in the data set whereas there are only 3 arguments assigned to key points 3 and 7 each. These key points could be treated as outliers or noise due to how few arguments align with them compared to others in the data set. This could be seen as removing misinformation about the topic, with misinformation being identified as claims or arguments made by a small number of sources. While there is a possibility of removing factually correct and important arguments from the data set with this approach, it is important to recognise that it is more important to maintain integrity and only show arguments which are sufficiently referenced by many sources. Due to this, it is likely that any clustering method and later summarization method will eject arguments from key points 3 and 7 and the final summary will cease to include these.

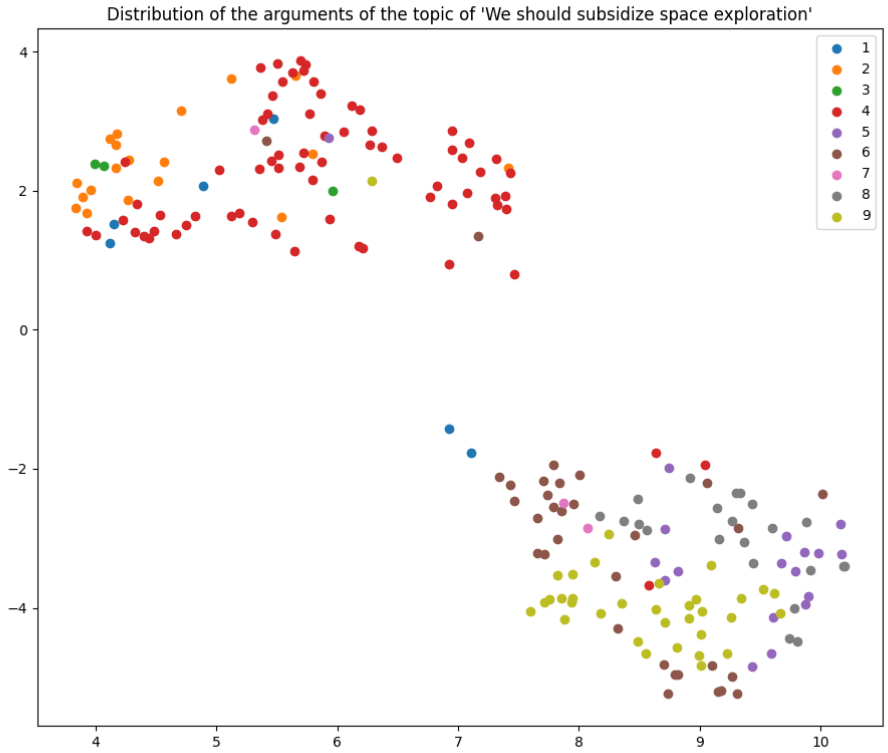

Figure 3.1: Distribution of the arguments of the topic of 'We should subsidize space exploration'

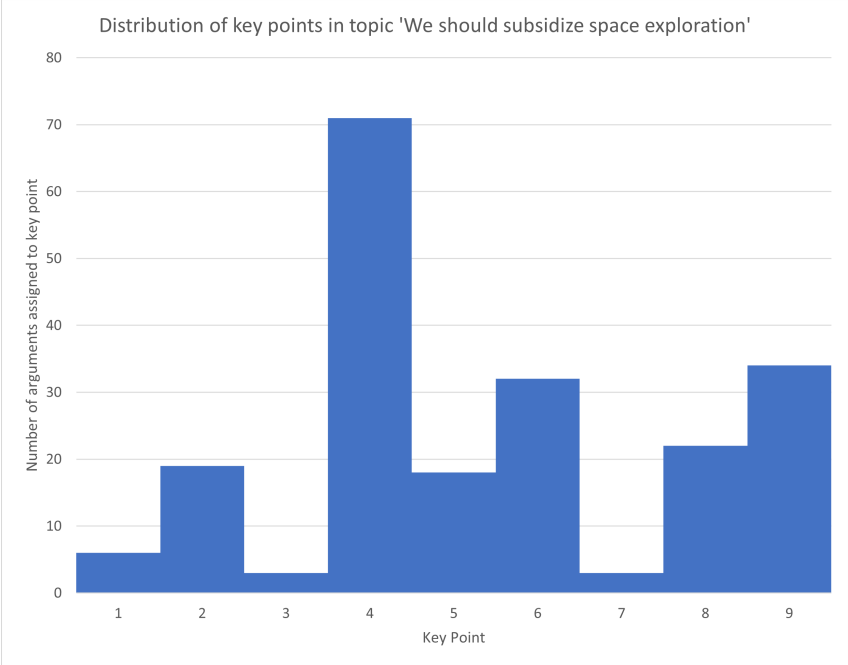

Figure 3.2: Distribution of the key points in the topic of 'We should subsidize space exploration'

3.3.1 KMeans

I conducted an experiment to evaluate the performance of KMeans clustering on the arguments of topic 22 in the data set, 'We should subsidize space exploration'. Sentence embeddings for each argument in this topic were generated and used as input to the clustering algorithm. While we can see that the data set has these arguments assigned to 9 different key points, we cannot assume knowledge of this, especially if it is to be used with other data sets and be a generalised solution for the automatic argument summarization problem. Therefore, one solution is to run the algorithm for many values of K and calculate the silhouette score for each resulting clustering, comparing the scores to try to determine the optimal number of clusters for the data. I experimented with the value of K ranging from 4 to 12 and table 3.2 shows the precision, recall, F1 score and silhouette score computed for each clustering the algorithm returned. We can draw many conclusions by observing the data collected in this experiment.

Firstly, while the highest silhouette scoring value for K resulted in the highest F1 score, we can see that the silhouette score does not map perfectly to the F1 scores achieved. For example, a value of 12 for K, resulted in a silhouette score of 0.0601 and a F1 score of 0.1759 while a value of 7 for K, resulted in a silhouette score of 0.0614 and a F1 score of 0.2412. While the silhouette scores for both values of K are very similar, a value of 7 for K resulted in a higher F1 scoring, with both higher precision and recall values. This suggests that comparing silhouette scores may not be the most accurate way of determining the optimal number of clusters for a given input of arguments.

Secondly, this experiment found that the KMeans algorithm resulted in the highest F1 score when returning all arguments clustered into 4 groups. Given that the gold standard to reach is 9 key points for this topic, I can conclude that the algorithm clusters arguments relating to different key points together. This is far from an optimal result and cannot be used because it will lead to the final summary missing out over half of the key points made in the data set. When the algorithm returned 9 clusters, the precision was 0.4929 and the recall was 0.1255, both of which are far too low to suggest a good clustering result. The low precision indicates that clusters are not very pure and further evidences that clusters contain multiple key points within them.

The poor performance of this algorithm can be attributed to the Euclidean distance between embeddings used to determine similarity. As in the background section, the Euclidean distance only considers the magnitude of the distance vector and therefore does not account for the direction which represents semantic similarity. This reduces the effectiveness of this algorithm because arguments are clustered based on their textual similarities such as the words and phrases they contain instead of whether they make the same or similar claim. I think this greatly limits this algorithm for this project as using the Euclidean distance alone is not accurate enough to cluster arguments which are so similar in content.

While not demonstrated in this experiment, one issue with KMeans clustering is its inability to deal with outliers or noise in the data set. Arguments not corresponding to any key points have been removed for testing purposes and to better evaluate clustering methods however it is important to recognise how the algorithm would perform differently if they were added back or a different data set was used. The issue is that the algorithm assigns every value in the data set into a cluster and outliers can drastically affect the cluster centres generated, making the algorithm less effective and returning a worse clustering of the data. A solution to this problem would be to identify outliers in the data set and remove them before clustering. However, this leads to the problem of outlier detection which risks the removal of important arguments but as discussed above it is important that the final summary generated only includes key points from arguments referenced by many sources.

Table 3.2: Evaluation metrics of KMeans clustering as K ranges from 4 to 12

K	Silhouette Score	Precision	Recall	F1 Score
4	0.1064	0.4160	0.3194	0.3614
5	0.0671	0.4320	0.2676	0.3305
6	0.0657	0.4352	0.2296	0.3006
7	0.0614	0.4052	0.1717	0.2412
8	0.0493	0.4568	0.1707	0.2485
9	0.0550	0.4929	0.1255	0.2001
10	0.0514	0.4817	0.1288	0.2032
11	0.0435	0.4826	0.1229	0.1959
12	0.0601	0.4830	0.1075	0.1759

In conclusion, this experiment highlights the main issues with KMeans clustering. Firstly, the number of clusters needs to be estimated by running the algorithm multiple times with different values however it is seen that silhouette scores do not translate perfectly to higher F1 scores and therefore the optimal number is not guaranteed to be selected. Secondly, this algorithm is ineffective for this data because it uses the Euclidean distance to determine similarity and therefore returns clusters which contain arguments assigned to multiple different key points.

3.3.2 Agglomerative Clustering

I conducted an experiment to evaluate the performance of Agglomerative Clustering on the arguments of topic 22 in the data set, 'We should subsidize space exploration'. This algorithm allows for many parameters to be adjusted and the effect of this has on the result will be investigated below.

Firstly, I investigated the effect different merging strategies have on the result of Agglomerative clustering when using Euclidean distance. For each merging strategy, the algorithm was ran with thresholds ranging from 5 to 25 and the F1 scores recorded to see which combinations resulted in the best performance. The results of testing ward, complete, average and single linkage merging strategies are shown in figure 3.3. All strategies converge to the same F1 score of 0.3211 and this is when the algorithm returns one singular cluster containing all the arguments in the topic. The threshold at which this occurs differs between the strategies and it is when the threshold allows the algorithm to merge every single cluster together in line with the selected strategy. The best F1 score of 0.4667 was achieved by using ward linkage and a threshold between 16 and 22 however it is important to mention that this resulted in the algorithm returning two clusters. This is simply too few and while the F1 score is relatively high compared to results when using KMeans, it does not necessarily indicate a good result. This teaches us that we cannot draw conclusions from comparing a single value or even 3 values in this case. The aim of the clustering method is to try to identify as many key points from the data set without compromising too much on recall and the number of duplicate key points in the final summary. Therefore when comparing any approaches to a task, the aim of the task has to be identified and as such the results you are most interested in. From the results of the experiment, I can conclude that ward linkage offers the best performance when Euclidean distance is used as the distance measure between embeddings. I think this is because ward aims to minimize the intra-cluster variance when merging and this results in clusters which are better defined compared to the other strategies.

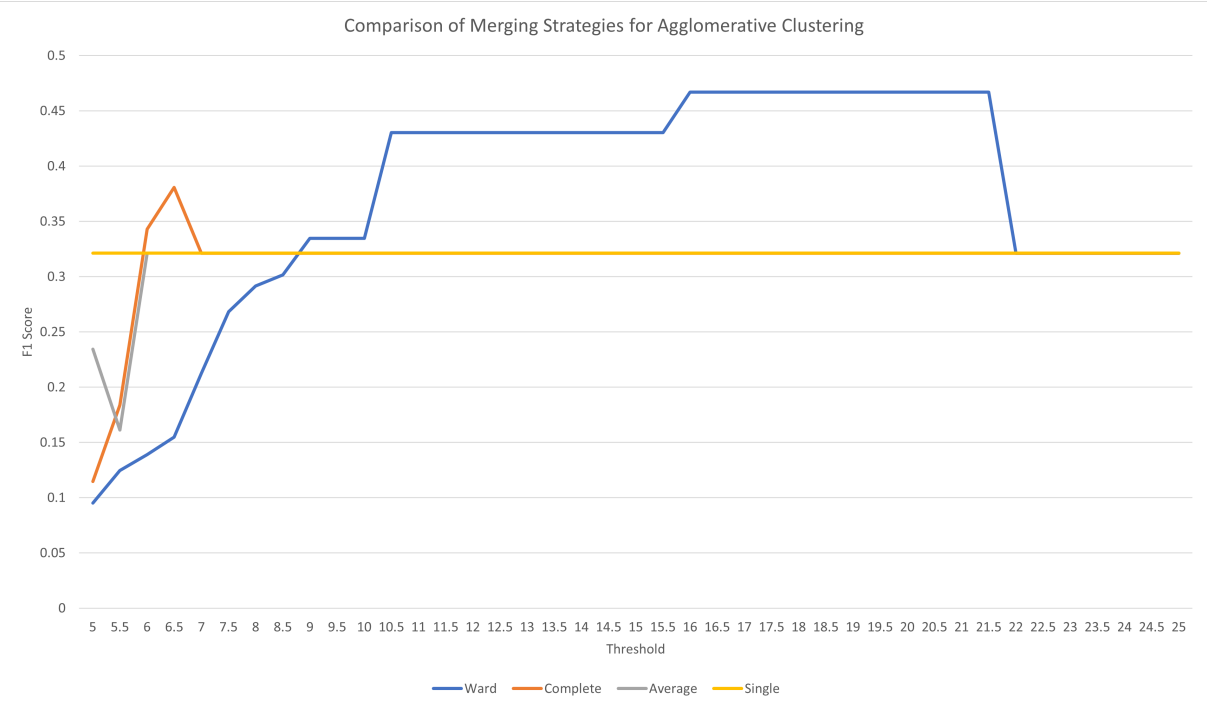

Figure 3.3: Comparison of F1 scores achieved by different merging strategies for Agglomerative clustering using Euclidean distance for the topic of 'We should subsidize space exploration'

The second experiment investigated the performance difference between using Euclidean distance and cosine similarity to compare sentence embeddings. The prior experiment was repeated using cosine similarity instead of Euclidean distance as comparison between embeddings and the results are shown in figure 3.4. For each merging strategy, the algorithm was ran with thresholds ranging from 0 to 1 as per the confines of the cosine similarity and the F1 scores recorded. Ward linkage is not compatible with using cosine similarity because it is not a distance measure and as such the variance within clusters cannot be calculated. The highest F1 score of 0.5415 was achieved using single linkage and a threshold of 0.5. While this spike in performance looks promising, examining the clusters returned when using these parameters shows that the high score is caused by 2 clusters, one containing a single argument with precision 1 and the other cluster containing all the other arguments in the topic with a recall of close to 1. When the average precision and recall across clusters is calculated, the values are skewed upwards which in turn causes the F1 score to be artificially inflated and not representative of the performance of the clustering. Single and average linkage strategies both converge to the same F1 score of 0.3211, exactly the same as in the previous experiment however complete linkage converges to a slightly higher score of 0.3659 where the algorithm returns 2 clusters instead of 1. Ward linkage generated the highest F1 score of 0.4667 when using Euclidean distance and this experiment found that a F1 score of 0.5415 could be achieved by using single linkage and cosine similarity. It is important to mention that both of these scores do not correspond to the algorithm clustering the data well as they both return all the arguments from the topic in just two clusters. Switching to using cosine similarity with Agglomerative clustering did not yield the performance improvements I thought it would and I think this is because ward linkage could not be used with cosine similarity. Considering this, I conclude that using Euclidean distance with ward linkage provides the best performance from this algorithm however it does not perform sufficiently well to be used as part of the system being developed in this project.

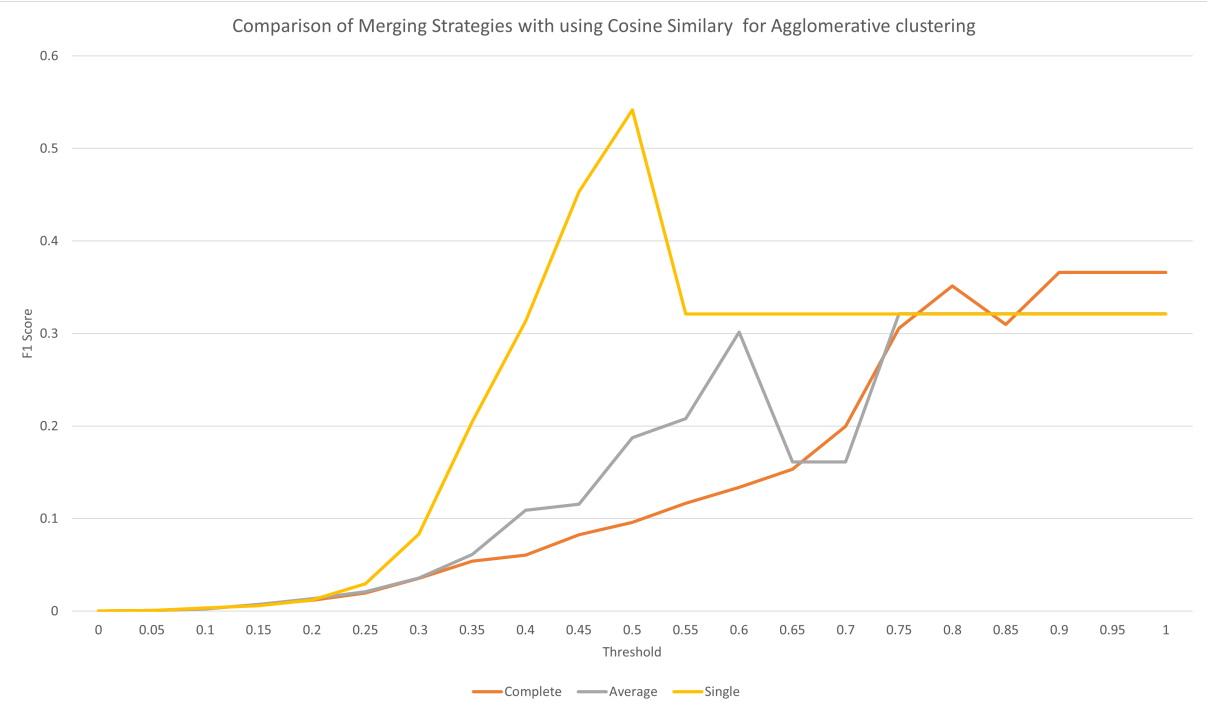

Figure 3.4: Comparison of F1 scores achieved by different merging strategies for Agglomerative clustering using cosine similarity for the topic of 'We should subsidize space exploration'

One potential issue which could be faced when clustering using Euclidean distance is that the optimal distance threshold may vary from topic to topic depending on how spread out sentence embeddings are in the vector space. This could be counteracted by running the algorithm with many different threshold values and calculating the silhouette score for each resulting clustering. However, as seen above, the silhouette score is not a perfect mapping to the F1 score and it is possible that the optimal threshold will not be found.

In conclusion, the two experiments on evaluating the performance of Agglomerative clustering can teach us two things. Firstly, we see that we cannot base the performance of an algorithm using certain parameters purely on the F1 score it achieves. The number of clusters the algorithm returns is very important as are the precision and recall values for each of them. Secondly, we see that using a better method of comparing embeddings, cosine similarity over Euclidean distance, does not guarantee better performance of the algorithm as there are other factors to take into account, in this instance the linkage methods available to use with each.

3.3.3 HDBSCAN

This algorithm differs from the previous two in a few ways but notably, by design it should be better at dealing with outliers and noise in the data set because it is not required to assign every data point to a cluster, instead it can choose to omit and classify them as noise. The number of arguments assigned to key points in the data set have a large variance with some key points having a small number of arguments related to them while others having substantially more. I think this may cause an issue for this algorithm as it could appear that there are 2 or 3 areas of high density in the vector space containing the sentence embeddings and treat all other arguments as noise. I conducted an experiment evaluating the performance of the HDBSCAN algorithm on the arguments of topic 22 in the data set, 'We should subsidize space exploration'.

The algorithm was run multiple times testing with setting the minimum cluster size from 2 through to 12 and the results can be seen in table 3.3, showing the precision, recall and F1 scores calculated for each resulting clustering. Setting the minimum cluster size to 11 and above, resulted in the algorithm not returning any clusters and classifying them all as noise. The highest F1 score of 0.5908 was achieved

by setting the minimum cluster size to 8 however the algorithm returned only two clusters, containing only 56 out of a total 205 arguments. Interestingly, the algorithm did not return more than 3 clusters for any minimum cluster size tested. This seems to suggest that the algorithm is incorrectly combining arguments from multiple key points into the same cluster. Looking at the clustering result evidences this claim and shows that the algorithm splits the arguments by stance perfectly and both clusters have a recall of 1. However, increasing the minimum cluster size directly correlates to more arguments being classified as noise by the algorithm instead of providing performance improvements or returning more or fewer clusters.

Table 3.3: Evaluation metrics of HDBSCAN whilst varying the minimum cluster size

Minimum Cluster Size	Precision	Recall	F1 Score
2	0.5905	0.4766	0.5275
3	0.5907	0.4764	0.5274
4	0.5682	0.5842	0.5761
5	0.5497	0.5800	0.5645
6	0.5462	0.5028	0.5236
7	0.2985	0.5605	0.3896
8	0.4193	1.0000	0.5908
9	0.4026	1.0000	0.5741
10	0.3673	1.0000	0.5373
11	0	0	0
12	0	0	0

In conclusion, this experiment highlights the ineffectiveness of this algorithm for clustering the sentence embeddings generated for each topic in the data set. I think I was right in my prediction that there would be an issue regarding the algorithm only being able to identify 2 or 3 areas of high density in the vector space. This is an issue with this algorithm however it is also an issue with the SBERT model not generating sentence embeddings distinct enough to allow this algorithm to effectively return well defined clusters.

3.3.4 Community Detection

Community detection uses cosine similarity to compare sentence embeddings and requires two parameters, a minimum community size and a threshold. I conducted an experiment to evaluate the performance of the Community detection algorithm, varying the threshold and minimum community size to identify the combination of parameters which yield the best results. The previous experiments focused on calculating and comparing the precision, recall and F1 scores to determine which parameters yield the best results. While these metrics are useful, this experiment will focus on the number of key points the algorithm is able to identify and the number of clusters it returns. Community detection returns clusters as well as the cluster centres and for this experiment, I will classify a key point as identified if there is a cluster centre which is assigned to that key point in the data set. Minimum community size was tested with values 2 through to 5 and for each the algorithm was ran with thresholds ranging from 0.5 to 1, recording the number of key points identified and the number of clusters returned each time. Due to the distribution of key points in this topic, as seen in figure 3.2, some key points have very few arguments assigned to them relatively and therefore it can be expected that as many as 3 key points could be classified as noise or outliers by the algorithm. Therefore a good solution would require the algorithm to identify at least 6 key points in this topic.

The results of the experiment can be seen in figure 3.5 where each bar represents results from a different minimum community size and error bars illustrate the total number of clusters returned by the algorithm. The combination of a minimum community size of 2 and a threshold of 0.825 identified the most key points in the topic, 8 out of 9, however it also returned 20 clusters. This is by far the best clustering algorithm tested so far in spite of the large number of clusters returned by it. A minimum community size of 4 with the same threshold of 0.825 identifies 6 key points which passes the minimum requirement set out above and only has 9 clusters compared to 20. This experiment clearly shows the Precision-Recall trade off and the need to prioritise one over the other when selecting which parameters to use. I think more information is required to make an informed decision about which combination of parameters to choose as the best performing for this algorithm.

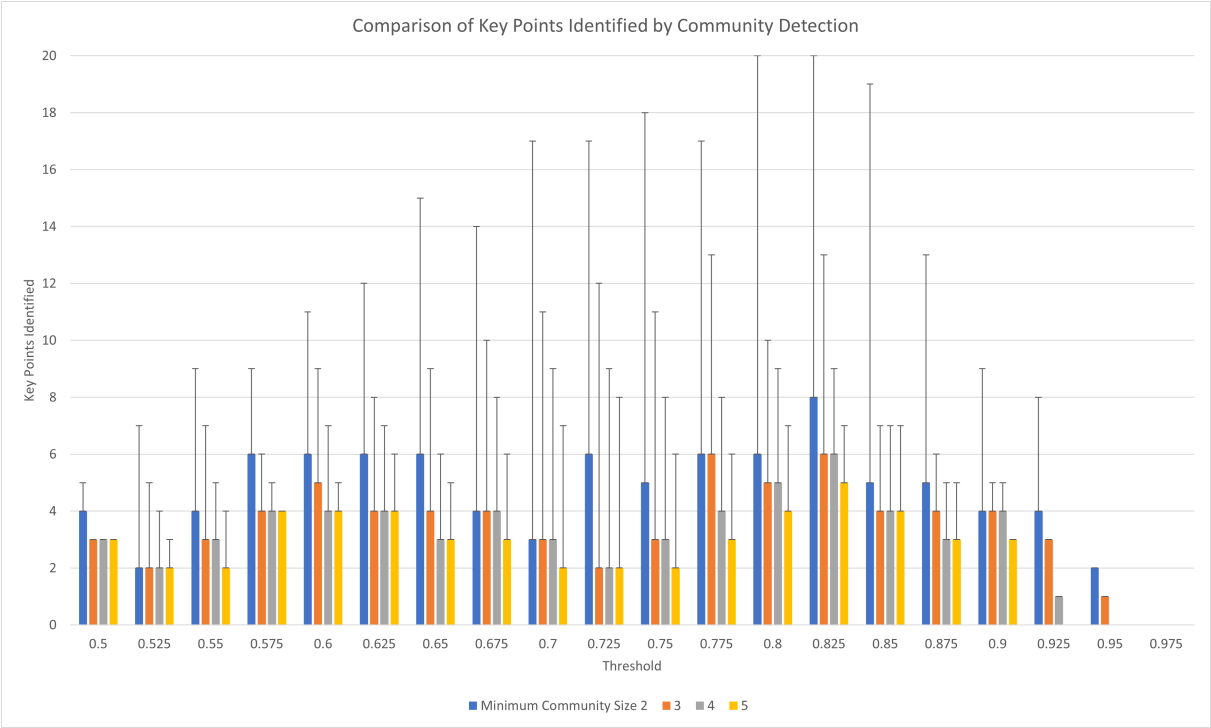

Figure 3.5: Comparison of key points identified using Community Detection for the topic of 'We should subsidize space exploration'

Following this, I conducted a second experiment to compare the performance of a select few parameter combinations across all topics in the data set using Community Detection. The experiment tests combinations of a threshold of 0.8 and 0.825 and minimum community size of 2 and 4, and records the number of key points identified for each topic and the number of clusters the algorithm returns for them. The result of this experiment can be seen in figure 3.6 where each bar represents results from parameter combinations and error bars illustrate the total number of clusters returned by the algorithm. We can see that a minimum community size of 2 and a threshold of 0.8 consistently returns 10 or more clusters per topic and 20 or more clusters for the majority of the topics. As clearly seen in the figure, the number of clusters returned by this combination of parameters is at least double the number of key points identified for all topics in the data set. Keeping the same minimum community size but using a threshold of 0.825, on average identifies the same number of key points and slightly reduces the number of clusters, however not substantially. Due to this, I consider this combination to perform better. While the number of clusters returned is still quite large relatively, it is an improvement. Setting the minimum community size to 4 caused a significant reduction in the number of key points identified and therefore cannot be considered as a viable parameter selection.

One of the key issues faced when working with this data set is that there are 24 topics and a method needs to be created which works well across all of these. The threshold value needs to stay consistent between all topics because there needs to be a standard minimum similarity for arguments to be clustered together and for key points to be identified. This value cannot vary from topic to topic because while it will provide better clustering results, it will allow this software to be easily manipulated. For example, it could cause arguments, from questionable sources which should be treated as outliers, be clustered along with arguments from legitimate sources and altering the final summaries generated by the system.

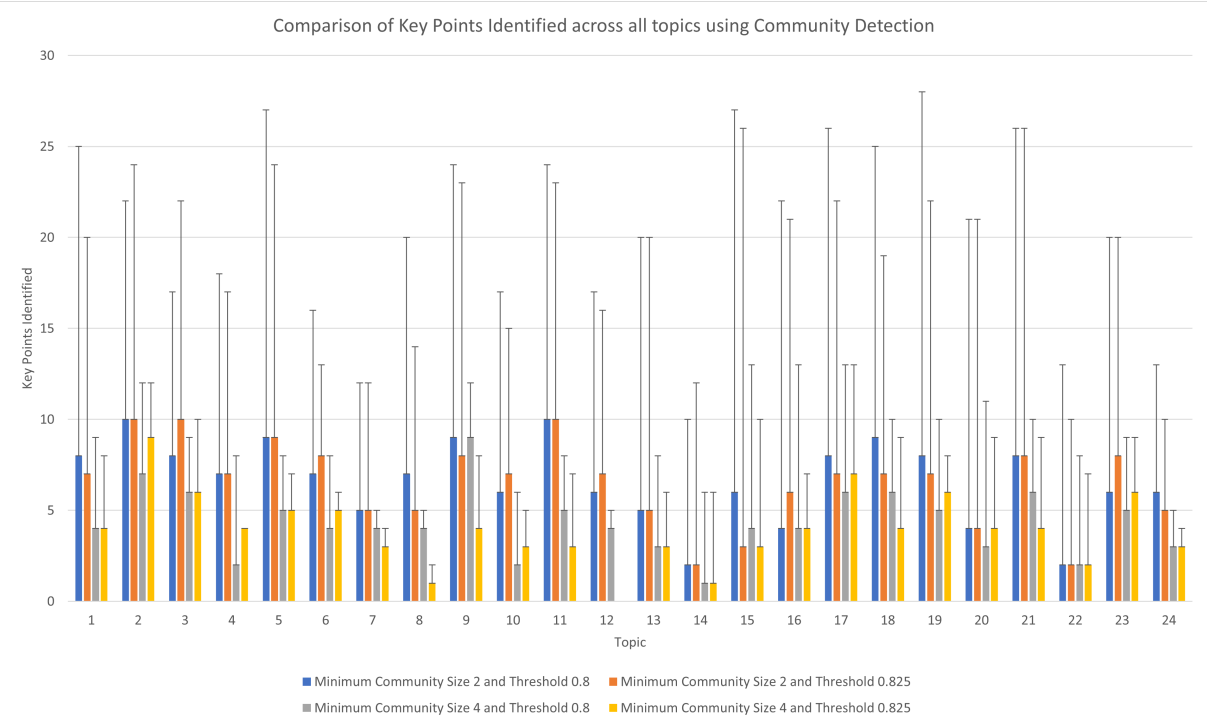

Figure 3.6: Comparison of key points identified across all topics in the data set using Community Detection

In conclusion, this experiment found the best parameters to use with the Community Detection algorithm are a minimum community size of 2 and a threshold of 0.825. This attempts to strike a balance between maximising the number of key points identified whilst minimising the number of redundant clusters returned by the algorithm. For this project, precision is valued more than recall as the intention is to capture as much of the essence of all the arguments as possible and provide a summary which represents them accurately and effectively. However there is a minimum recall required otherwise the summary risks becoming too long and not concise enough to act as an actual summary of the arguments in the topic. This is why these parameters were selected for this algorithm and while the resulting clustering is not perfect due to returning a larger number of clusters than necessary, it is satisfactory and should allow good summaries to be generated from them.

3.3.5 Hierarchical KMeans Clustering

KMeans clustering faced many issues mainly that it would assign all data points to a cluster without considering if they could be outliers, and it was difficult to estimate the optimal number of clusters for the data to pass in as the value of K. The main issue with Agglomerative clustering was the inability to use ward linkage as the merging strategy when using cosine similarity to compare the embeddings with each other. This meant that the best result the algorithm achieved was using the Euclidean distance and I believe it had the potential to deliver a great clustering result if ward linkage could be used with cosine similarity. Considering the issues with these algorithms, I propose a novel algorithm, taking inspiration from both of these clustering methods, which aims to address the issues faced by them by adopting a hierarchical approach where all arguments start in their own cluster and through iterations of the algorithm, cluster centres are recalculated, merged together and arguments assigned to them. The algorithm takes a list of arguments as input as well as two parameters, the minimum and maximum similarity threshold. The minimum similarity threshold determines at least how similar an argument has to be to a cluster centre to be assigned to it while the maximum similarity threshold determines how similar two cluster centres need to be to be merged into one. The psuedocode for this novel algorithm is shown in Algorithm 3.1.

Algorithm 3.1: Hierarchical KMeans Algorithm

```
Data: arguments, minimum_threshold, maximum_threshold
Result: Dictionary of Lists, clusters, where each list contains all the arguments in the cluster
clusters ←Empty Dictionary
foreach arg in arguments do
    c ←arg + two  most similar arguments
    insert c into clusters
end
while cluster_centres are volatile do
    cluster_centres ←find most central argument in each cluster
    /* Find pairs of cluster centres which are at least as similar as the threshold and only keep
       the first centre from each pair                                                          */
    foreach c in cluster_centres do
        closest_centre ←find most similar argument in cluster_centres
        sim ←similarity between c and closest_centre
        if sim ≥maximum_threshold then
            remove closest_centre from cluster_centres
        end
    end
    /* Assigning each argument to its most similar cluster centre if above the threshold        */
    foreach arg in arguments do
        closest_centre ←find most similar argument in cluster_centres
        sim ←similarity between arg and closest_centre
        if sim ≥minimum_threshold then
            append arg to clusters[closest_centre] end
        end
        /* Merge clusters of size 1 or 2 if similarity is above the threshold                   */
        foreach c in cluster_centres do
            if Length of c = 1 or 2 then
                closest_centre ←find most similar argument in cluster_centres
                sim ←similarity between c and closest_centre
                if sim ≥minimum_threshold then
                    merge closest_centre and c
                end
            end
        end
        /* Remove any clusters containing 2 or fewer arguments                                  */
        foreach c in cluster_centres do
            if Length of c ≤ 2 then
                remove c from clusters
            end
        end
    end
```

I conducted an experiment to evaluate the performance of my algorithm clustering the arguments of topic 22 in the data set, 'We should subsidize space exploration' and to identify the optimal minimum and maximum threshold values to use for the data set. For this experiment, key point identification is defined as there existing a cluster centre which is a member of that key point. This is to ensure that the key point has actually been identified and not just been mis-clustered and is in the data set in general. I tested the algorithm with minimum and maximum threshold values ranging from 0.5 to 1 and recorded the number of key points each assignment was able to identify. The results of the experiment can be seen in figure 3.7. The algorithm was able to identify 8 out of 9 key points in the topic and this was achieved when the minimum similarity threshold was between 0.5 and 0.65 and the maximum similarity threshold was between 0.85 and 0.95. This figure does not include the number of clusters returned by the algorithm for each of these parameter assignments and this is important to review and consider before selecting the most optimal parameters to use with this algorithm and data set.

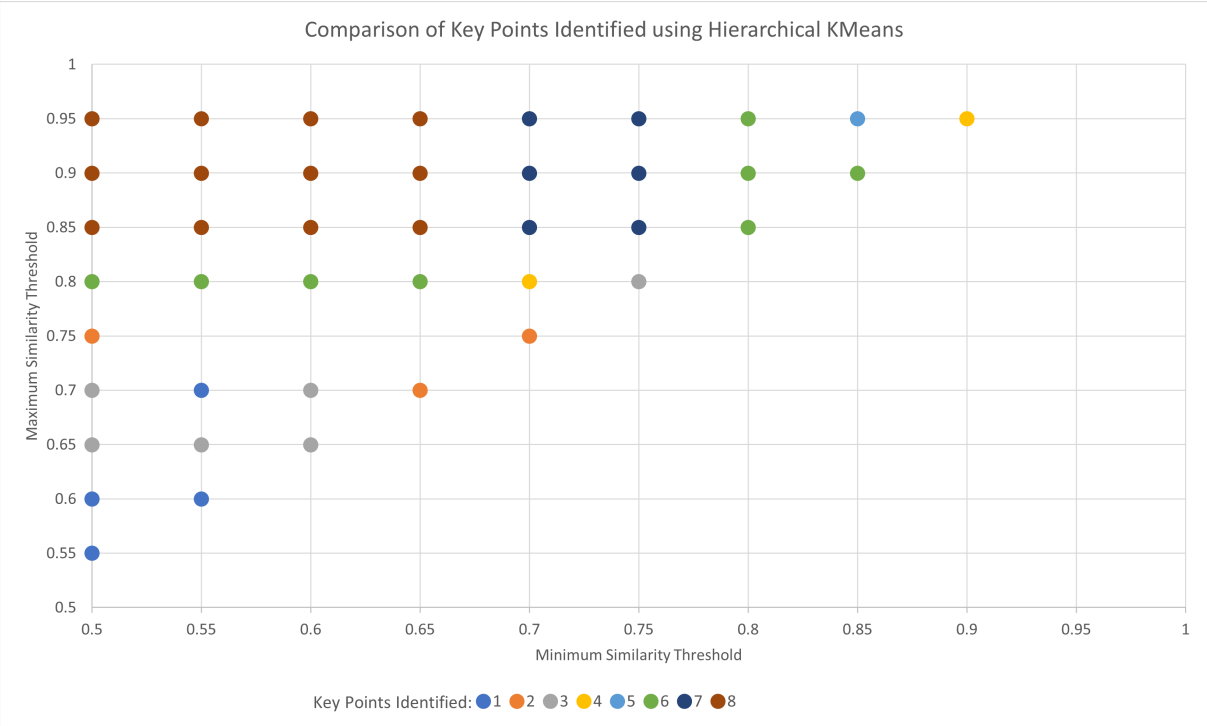

Figure 3.7: Comparison of the number of key points identified using Hierarchical KMeans for the topic of 'We should subsidize space exploration'

Table 3.4 shows the number of clusters the algorithm returns for each combination of minimum and maximum similarity thresholds between 0.5 and 1. By cross referencing this table with the above figure, We can see that in order to identify 8 key points from the topic, the algorithm will return a minimum of 24 clusters, using 0.65 and 0.85 as the respective thresholds. Comparing this to Community Detection, which identified 8 key points for this topic and returned 20 clusters, it performs slightly worse down to the extra 4 clusters returned with this method. However, where this algorithm excels is its adjustablity. We have the choice of reducing the precision and identifying one key point fewer in exchange for returning 15 clusters instead, 9 less, by changing the thresholds to 0.75 and 0.85 respectfully. I think that this is the best result this algorithm is capable of because in order for it to return fewer clusters, there would have to be a huge sacrifice in the number of key points identified rendering any improvements in recall useless. While the algorithm returning 15 clusters is not ideal, over double the key points it identifies, it is possible that this number could be reduced during the summarization step of the project. However, even it doesn't, I believe it is better for the summary to contain duplicate or similar key points as opposed to those key points not being part of the summary at all.

Table 3.4: Number of clusters returned by Hierarchical KMeans for the topic of 'We should subsidize space exploration' for all combinations of threshold values between 0.5 and 1

	0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95
0.5		2	2	4	3	6	15	27	39	44
0.55			2	4	4	6	14	25	34	43
0.6				4	6	6	15	27	35	44
0.65					4	6	13	24	34	41
0.7						4	7	21	32	37
0.75							5	15	27	36
0.8								14	22	26
0.85									14	14
0.9										6
0.95										

In conclusion, this experiment shows that my algorithm can compete in terms of performance to established methods such as Community Detection, and even surpass it. The algorithm does return more clusters than is necessary and it will be interesting to see how summarising methods will deal with this issue however most key points in the topic are identified using optimal threshold values determined in this experiment.

With that experiment finding the optimal minimum and maximum similarity threshold values to use with the algorithm, I conducted a second experiment to observe the performance of the algorithm across all the topics in the data set. For each topic, the algorithm was ran using a minimum similarity threshold of 0.75 and a maximum similarity threshold of 0.85. Figure 3.8 shows the the number of key points the algorithm identified for each topic, how many it failed to identify and the error bars illustrate the number of clusters the algorithm returned for that topic. This result could definitely be improved as some topics, specifically topics 7, 11 and 23, are missing a large percentage of their key points. This algorithm has performed better on average than Community Detection when considering the number of key points identified for each topic. This algorithm also returns fewer clusters than Community Detection, for example, 19 clusters for topic 22 compared to 15 clusters using this algorithm.

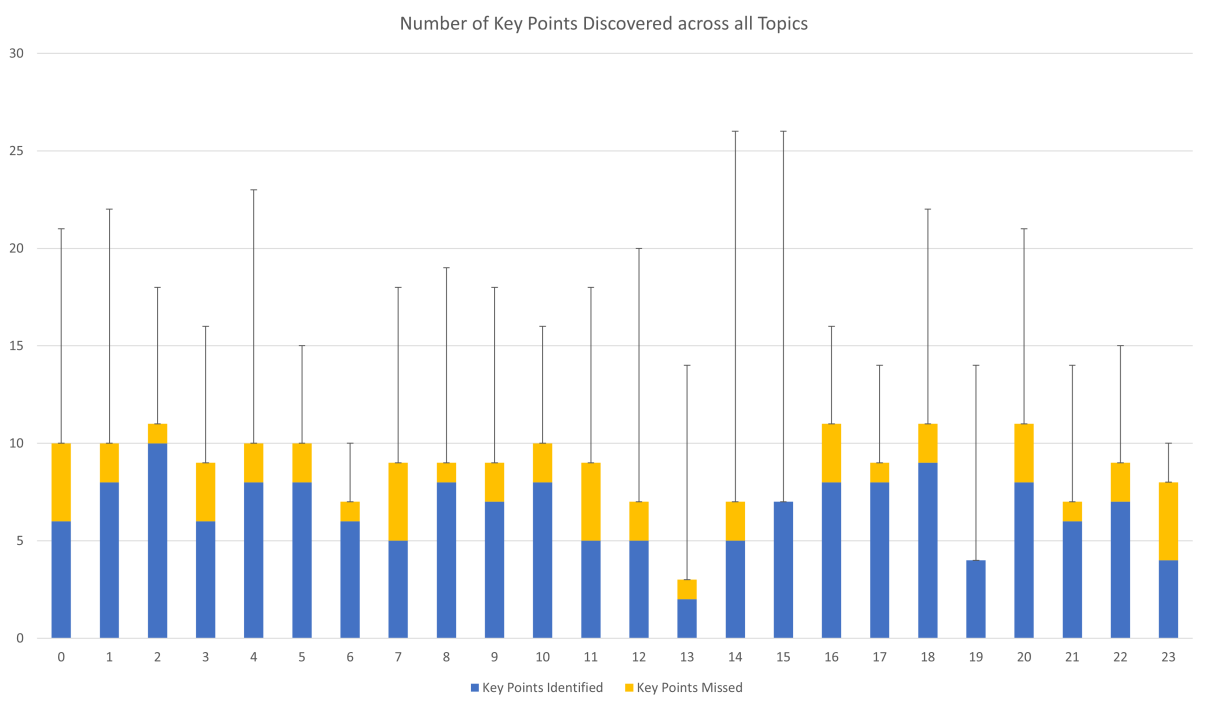

Figure 3.8: Comparison of the number of key points identified using Hierarchical KMeans for all topics in the data set

To conclude, Hierarchical KMeans offers the best performance for clustering the sentence embeddings generated in the data set compared to the established clustering algorithms available which have been tested in this paper. The main downside of this algorithm is the large number of clusters it returns, on average double the key points identified in the topic. This poses an issue because the summary is required to be short and concise, and duplicate key points will contradict this goal. Therefore the summarization part of this project will need to develop a method of identifying duplicate key points and removing them from the summary whilst ensuring that unique key points are not mis-classified in this process.

3.3.6 Greedy Nearest Neighbours Clustering

Hierarchical KMeans was shown to offer the best performance compared to all other algorithms tested however it was not without fault. While it managed to identify a large number of key points across topics, it would generate far too many clusters presenting a challenge for future summarization. I propose another novel algorithm, Greedy Nearest Neighbours, which aims to able to reduce the number

of clusters returned compared to Hierarchical KMeans whilst maintaining or increasing the number of key points identified. This algorithm adopts a greedy approach where clusters are created by identifying the largest list of similar arguments based on a threshold until no more lists of 2 or greater arguments can be found and the algorithm returns the clusters it has found. In theory this algorithm has the benefit of simplifying the task after each iteration and I think this should allow the algorithm to be more accurate. The psuedocode for this algorithm is shown below in Algorithm 3.2.

Algorithm 3.2: Greedy Nearest Neighbours Algorithm

```
Data: arguments, threshold
Result: List of Lists, clusters, where each list contains all the arguments in the cluster
clusters ←[]
while True do
    /* Identify largest list of similar arguments                                    */
    largest_list ←[]
    foreach arg1 in arguments do
        cluster ←[]
        foreach arg2 in arguments do
            sim ←similarity between arg1 and arg2
            if sim ≥threshold then
                append arg2 to cluster
            end
        end
        if Length of cluster > Length of largest_list then
            largest_list ←cluster
        end
    end
    /* Quit algorithm when there are no longer lists of similar arguments of size 2 or greater  */
    if Length of largest_list < 2 then
        break
    end
    /* Creating cluster from largest list of similar arguments found                 */
    append largest_list to clusters
    remove all largest_list from arguments
end
```

I conducted an experiment to evaluate the performance of this algorithm whilst varying the threshold value in order to identify the optimal value across all topics in the data set. This is a similar experiment to the one for Hierarchical KMeans algorithm and key point identification is defined the same; as there existing a cluster centre which is a member of that key point. I tested the algorithm with the threshold ranging from 0.8 to 0.9 and recorded the average key point coverage (the percentage of key points identified out of the number in the data set) and the percentage of summaries returned (the ratio between the number of summaries and the number of key points in the data set). The results of this experiment can be seen in figure 3.2. A threshold of 0.81 results with the highest percentage of key points identified, 77%, however the algorithm also returns on average double the number of summaries as key points in the data set. A threshold of 0.82, reduces the key point percentage to 75% and also reduces the percentage of summaries returned from 200% to 188%. I think that a threshold of 0.81 is the optimal value as it allows the algorithm to on average identify the largest number of key points from the data set.

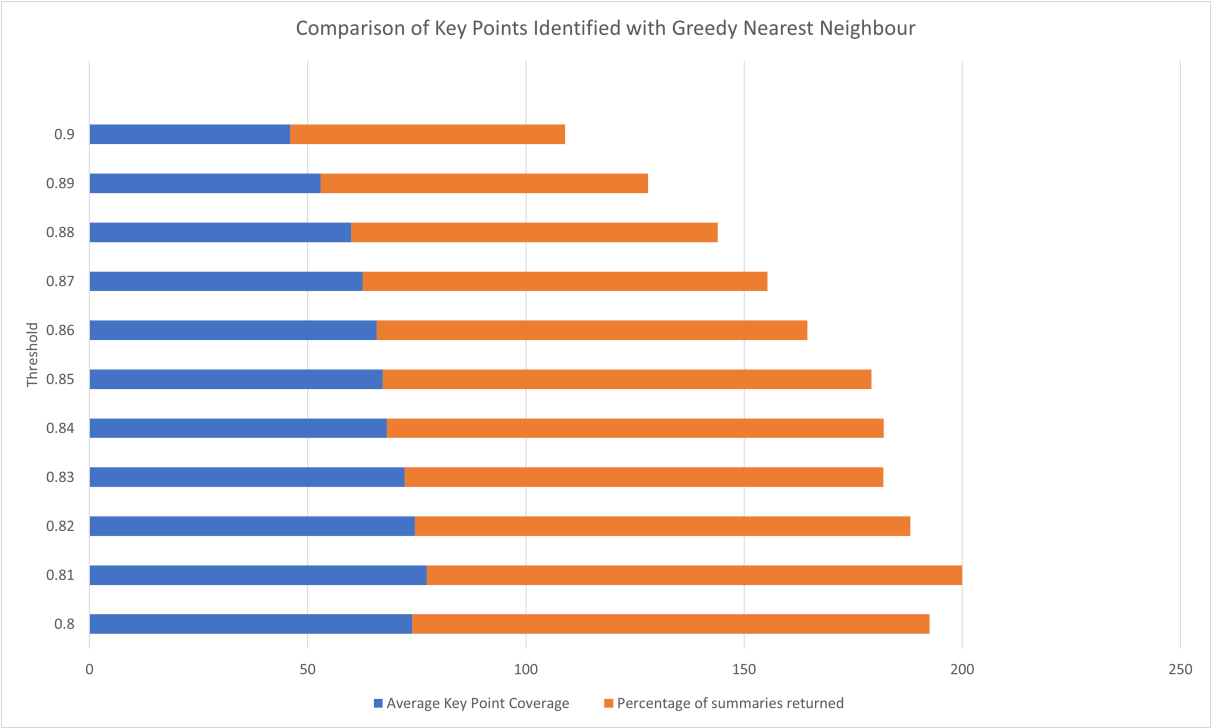

Figure 3.9: Comparison of Key points identified with Greedy Nearest Neighbour algorithm when varying the threshold value

To conclude, this experiment found that this algorithm performs very well at identifying key points within topics in the data set however falls short in regards to the number of clusters it returns. The algorithm suffers from the same issue as Hierarchical KMeans and similarly also on average returns double the number of clusters as key points in the topic.Therefore it will be interesting to see how summarization methods handle this issue and if duplicate key points can be merged. It seems like the performance of this algorithm is very similar to Hierarchical KMeans and therefore it will be interesting to compare them with each other more closely in a later experiment.

3.4 Analysing Summary Quality

There are multiple approaches to generating summaries from a cluster of arguments and it is important that these summaries are concise and capture the key point the arguments are trying to make. I conducted an experiment to investigate how useful using ROUGE scores to analyse the quality of summaries is. Table 3.5 shows the ROUGE scores calculated when clustering the arguments in topic 22 based on their assigned key point in the data set and comparing them to it. ROUGE compares words, bi-grams and sequences of words to generate the scores and the key points in the data set have been written by an expert debater to be deliberately short and abstract in order to be able to encapsulate as many arguments as possible. This means that firstly there is a significant difference in word count between a given argument and its assigned key point which reduces the probably of matches between them and secondly the key point is phrased differently to the arguments whilst remaining semantic similar. ROUGE is unable to measure semantic matches, only textual and therefore the same words need to used to receive a high score. This suggests that looking at the ROUGE score alone will not be enough to evaluate the quality of a generated summary especially for abstractive summarization where the goal is to generate summaries very similar to the ones in the data set. All clusters have a higher ROUGE-1 score than ROUGE-2 and I think this is because every argument will contain the main topic word as will the key point, for example for topic 22, 'We should subsidize space exploration', all arguments will contain the word "space" and so will the key points of that topic. This guarantees at least one matching word when comparing arguments to key points whereas it is less likely that there are matching bi-grams.

Table 3.5: ROUGE scores evaluating data set provided key points for the topic of 'We should subsidize space exploration'

Cluster	Cluster Size	ROUGE-1	ROUGE-2	ROUGE-L
1	6	0.3566	0.1661	0.3433
2	19	0.2901	0.1523	0.2901
3	3	0.2368	0.0833	0.1740
4	71	0.2212	0.0814	0.1772
5	18	0.2375	0.0681	0.2125
6	32	0.2150	0.1038	0.2095
7	3	0.2039	0.0824	0.2039
8	22	0.2840	0.1194	0.2551
9	34	0.2478	0.1084	0.2281

To conclude, the experiment suggests that the ROUGE score may not be a good measure of summary quality for abstractive summarization methods however it should provide reasonable results on extractive methods. The main issue is that it only checks for matching words and sequences without considering the context in which they are in to inform the score.

After my experience with ROUGE scores, I conducted an experiment into how useful the BERTScore will be in analysing the quality of generated summaries. Table 3.6 shows the precision, recall and F1 scores calculated for each cluster using the BERTScore calculated when repeating the same experiment as above. These high scores appear to be more representative of the quality of the summary as less importance is placed on the words the key point consists of and more the meaning of it and specifically how similar to is to the argument the score is being calculated for. It could be agreed that the metrics calculated by BERTScore are not very intuitive and it is hard to see how they are calculated however I wouldn't class this as an issue because for most applications, the F1 score is computed and this is more intuitive, with higher F1 scores indicating higher similarity.

Table 3.6: BERT scores evaluating data set provided key points for the topic of 'We should subsidize space exploration'

Cluster	Cluster Size	Precision	Recall	F1 Score
1	6	0.9165	0.8860	0.9009
2	19	0.9263	0.8669	0.8955
3	3	0.8920	0.8649	0.8782
4	71	0.8765	0.8627	0.8695
5	18	0.8897	0.8784	0.8839
6	32	0.8931	0.8717	0.8822
7	3	0.9060	0.8590	0.8818
8	22	0.9176	0.8916	0.9043
9	34	0.8926	0.8579	0.8749

In conclusion, the experiment finds that the BERTScore provides a lot of value and insight into the quality of a generated summary as using a BERT model allows for greater and more accurate comparison of text. It also allows abstract summaries to be better compared and judged on their quality.

3.5 Comparing Summarisation Approaches

3.5.1 Extractive Summarisation

Most Central Argument

The initial approach to generating summaries for a cluster of arguments involved finding the argument which when compared with all other arguments in the cluster, was on average the most similar. For each argument, the sentence embedding was generated and the cosine similarity between itself and every other argument was averaged to find the average similarity the argument had across the cluster. The argument with the highest score can be seen as the cluster centre and most representative of the arguments within

it and as such it is chosen to be the summary of the cluster.

I conducted an experiment to investigate how well the Most Central Argument method works in generating extractive summaries of clusters for the topic of 'We should subsidize space exploration'. In this experiment, arguments were grouped based on the key point they are assigned to in the data set allowing the results to be independent on any clustering method it could be used with. Table 3.7 presents the key points in the data set side by side with the summaries generated by identifying the most central argument in each cluster. The most central argument is longer than the key points in the data set as the key points have been written by an expert debater and purposefully been made as short and concise as possible. This is one drawback of this method because one argument from each cluster has to be selected to act as the summary and it is not possible to reduce the length of this. In spite of this, the summaries generated are very similar to the key points in the data set and capture the essence of all the arguments within each cluster well.

For each generated summary, the BERTScore is calculated by comparing each argument in the cluster to it. This is similar to how the most central arguments are identified and I predict that the scores of the summaries will be relatively high due to this. Table 3.8 shows the BERT scores of the summaries generated when calculating the most central argument within each cluster and using it as the summary. The scores are on average slightly higher compared using the key points as summaries in the experiment above (table 3.6). This could be because the summary is a member of the cluster it is in and therefore when calculating the BERTScore of the cluster, it scores 1 for all metrics (precision, recall and F1 score), boosting the average.

Table 3.7: Comparison of data set key points and most central argument in each cluster for the topic of 'We should subsidize space exploration'

Cluster	Data Set Provided Key Point	Most Central Argument
1	Space exploration can be carried by the private sector	we should not subsidize space exploration because we already spend enough on it. we can always have private companies like elon musk pay for additional space exploration
2	Space exploration is expensive	we shouldn't subsidize space exploration because it can be very costly
3	Space exploration is ineffective	exploration of space has not produced any tangible benefits yet, so funding should be discontinued
4	There are issues more important to fund than space exploration	there are much more important things to spend money on; space exploration is a low priority and should not be subsidized
5	Space exploration can help in colonizing the resources/lands of other planets	we should subsidize space exploration because it could help find useful things needed on earth for the world
6	Space exploration improves science/technology	space exploration provides the basis for scientific development in many ways and should be subsidized
7	Space exploration is financially beneficial	space exploration has the potential to bring an economy worth trillions . we should do all in our power to help that become a reality.
8	Space exploration is necessary for the future survival of humanity	space exploration is vital to the future of humanity, when we destroy this planet, we will need to have an alternative
9	Space exploration unravels information about the universe	we should subsidize space exploration because it can help provide us with education of the universe around us

Table 3.8: BERT scores evaluating the summaries generated by finding the most central argument for the topic of 'We should subsidize space exploration'

Cluster	Precision	Recall	F1 Score
1	0.8891	0.9153	0.9019
2	0.9120	0.9001	0.9058
3	0.9304	0.9320	0.9312
4	0.8839	0.8865	0.8851
5	0.8737	0.8864	0.8800
6	0.8950	0.8961	0.8955
7	0.9108	0.9097	0.9102
8	0.8949	0.8999	0.8973
9	0.8990	0.8969	0.8979

In conclusion, the results of this experiment are independent of any clustering method and therefore the results seen show the potential of the summarization method if the clustering was perfect. While this experiment shows that the Most Central Argument method is effective in generating extractive summaries of clusters, it is difficult to predict if similar results will be seen when presented with clusters with lower precision and recall.

3.5.2 Abstractive Sumarisation

Google's Pegasus Model

I conducted an experiment to investigate the quality of the summaries generated by the Pegasus model for the topic of 'We should subsidize space exploration'. This experiment follows the same set up as above when testing the Most Central Argument method to ensure independence from clustering algorithms however uses the Pegasus model instead to generate a summary for each cluster. The model returns multiple summary candidates if the combined length of all the arguments within the cluster exceed a certain threshold. When this is the case, each summary candidate can be evaluated with the BERTScore to determine which one best represents the arguments in the cluster. Table 3.10 presents the key points in the data set side by side with the summaries generated by the Pegasus model. The model does a good job of generating these summaries which are very similar to the key points written by an expert debater. However, the summaries generated for clusters 1 and 3 are not representative of the arguments in them at all and this highlights one of the issues with abstractive summarization methods. These models are prone to errors because generating new text is a complex task and often the coherency of the output is prioritised over the accuracy of it.

Table 3.9: BERT scores evaluating summaries generated by Pegasus model for the topic of 'We should subsidize space exploration'

Cluster	Precision	Recall	F1 Score
1	0.8614	0.8667	0.8640
2	0.9164	0.8974	0.9067
3	0.8648	0.8729	0.8687
4	0.9093	0.8825	0.8957
5	0.9000	0.8815	0.8906
6	0.9055	0.8922	0.8987
7	0.9089	0.8795	0.8938
8	0.9015	0.9008	0.9011
9	0.9050	0.8951	0.9000

Table 3.10: Comparison of data set key points and summaries generated by Pegasus model for the topic of 'We should subsidize space exploration'

Cluster	Data Set Provided Key Point	Model Generated Summary
1	Space exploration can be carried by the private sector	What do you think should be done about space exploration?
2	Space exploration is expensive	government should not subsidise space exploration because it is too expensive.
3	Space exploration is ineffective	How much do you think India should spend on space research?
4	There are issues more important to fund than space exploration	space exploration is a waste of money and it would be better spent elsewhere.
5	Space exploration can help in colonizing the resources/lands of other planets	Subsidize space exploration because we need to find a new planet to live on.
6	Space exploration improves science/technology	space exploration should be subsidized as it brings innovation and scientific development.
7	Space exploration is financially beneficial	space exploration has the potential to bring an economy worth trillions.
8	Space exploration is necessary for the future survival of humanity	space exploration is vital to the future of humanity, when we destroy this planet, we will need to have an alternative.
9	Space exploration unravels information about the universe	We should subsidize space exploration as it is important for the world to know what else is out there.

For each generated summary, the BERTScore is calculated by comparing each argument in the cluster with itself and the results can be seen in table 3.9. The scores are comparable to the scores computed when using the key points as summaries in the experiment above (table 3.6). The scores show that clusters 1 and 3 are not as accurate as the rest of the summaries generated by the model which reflects what I observed. In conclusion, the results of this experiment are show the potential of the the Pegasus model has in generating summaries when presented with perfectly clustered arguments. The model generates very good summaries for most of the clusters however this is not guaranteed as seen for clusters 1 and 3. This method is not reliable enough to be used, especially as in this experiment, it was presented with perfectly clustered content. I think when given clusters with lower precision and recall, the model would be more inaccurate and therefore it cannot be considered as a viable summarization method for this project.

Model Hallucination

Ziwei Ji et al. define hallucination in the context of NLP as "the generated content that is nonsensical or unfaithful to the provided source content" [7]. This would be when the model will return something which does not seem justified from the provided inputs. This was seen in the above experiment where for clusters 1 and 3, the model generated a summary which was not at all representative of the arguments it was provided with. Another example is when the summarization model sometimes fabricates quotes from famous people or academics or invents figures. For example, for topic 22, key point 7 has 23 arguments assigned to it in the data set. Feeding these arguments into the Pegasus model generates the following list of plausible summaries:

Key Point 7: "Space exploration is necessary for the future survival of humanity"

Summaries:

- "Humanity must find a new home in space."
- "In our series of letters from African journalists we look at the importance of exploration."
- "BBC News takes a look back at some of the top stories of the week."
- "It's time to start thinking about space."
- "Space exploration is the only way to save the world."

- "BBC News takes a look at the future of space."
- "BBC News takes a look at the future of space exploration."
- "Scientists at the Massachusetts Institute of Technology (MIT) have come up with a novel way to deal with the threat of climate change."

The data set has no mention of the BBC or MIT or African journalists and the model has added these details in when generating the summaries. One explanation for this that the model has been trained with a huge number of articles, all of which will include quotes and talk about organisations and companies and when attempting to generate a summary for the input texts, it will use its training and past experiences to dictate how the summary is be generated. Hallucination poses an issue for this project because the aim is for these summaries to be accurate, concise and representative of the arguments in the data set and by introducing information not found in the data set, ruins the integrity of the summaries. Evaluating all summary candidates returned by the model and selecting the best scoring one can attempt to remedy this issue however as seen in the previous experiment and in table 3.10, this does not always work.

3.5.3 Identifying duplicate key points across summaries

Some clustering algorithms return twice or three times the number of clusters as actual key points in the data set. These algorithms are prioritising precision and identifying the largest number of key points they can, at the cost of recall and hence returning so many clusters. A summary is generated for each cluster and this leads to the final topic summary being too verbose and repeating itself in places. For this reason, a method needs to be developed to identify and remove duplicate key points within the list of summaries generated using the methods described above.

The first approach is to generate sentence embeddings for each summary and using cosine similarity, identify summaries sufficiently similar to each other and remove all but one of them. For example, the Hierarchical KMeans algorithm will return 15 clusters, containing 7 unique key points, for topic 22 and the aim is to minimise this to as close to 7 key points as possible without compromising on accuracy.

I conducted an experiment comparing the effect different threshold values have for identifying duplicate key points in summaries. For each topic in the data set, I used the Hierarchical KMeans algorithm to cluster the arguments and the Most Central Argument method to generate summaries of them. With a list of summaries for each topic, different threshold values were tested and results were recorded. The aim of this experiment is to try to see if there is a threshold value which would reduce the number of summaries identified substantially without reducing the number of key points at all not by very little. Figure 3.10 shows how different threshold values affect the average key point coverage (the percentage of key points identified out of the number in the data set) and the percentage of summaries returned (the ratio between the number of summaries and the number of key points in the data set). A threshold value of 0.85 returns the summaries unchanged and this is because it is the same threshold value which was used with the Hierarchical KMeans clustering to generate the clusters. Considering these results, I do not think that this method produces any practical results as when the number of summaries is reduced to a reasonable amount for example 103% of the number of key points for that topic, these summaries only represent 55% of the key points in the data set. This is far too low and cannot be used for this project as close to half of the main points are missed from the final summary.

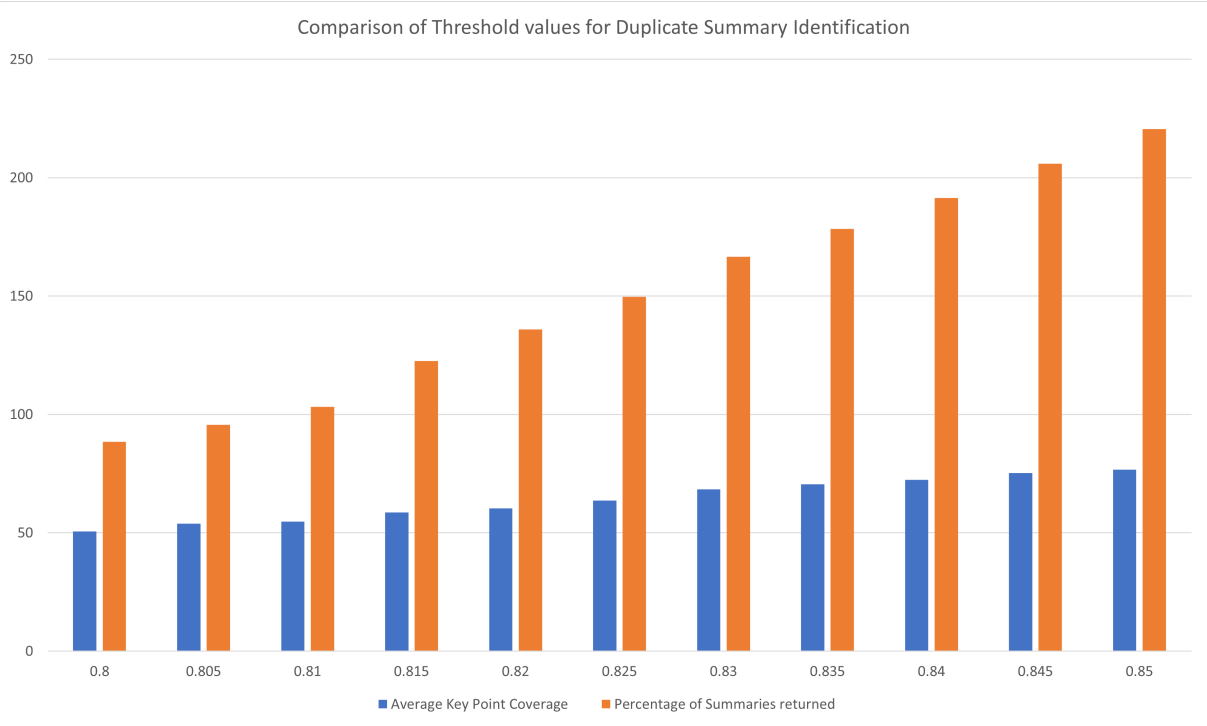

Figure 3.10: Comparison of threshold values for duplicate summary identification across all topics in the data set

I conclude that there is too large of a trade off between reducing the number of summaries and the key points they represent. Therefore, I decide that this method cannot be used in this project as accuracy is valued more than conciseness.

Chapter 4

Critical Evaluation

This section will detail the final system developed and explain the rationale behind decisions taken in the development process. The SBERT model chosen for this project was "paraphrase-multilingual-MiniLM-L12-v2" because the experiment (see table 3.1) comparing the 5 highest rated models found it delivers the greatest performance for the data set used in project. Popular and standard clustering algorithms were tested extensively in the previous section and the analysis of them influenced the design and development of a new clustering algorithm, Hierarchical KMeans and Greedy Nearest Neighbour. The former aimed to combine the benefits of Agglomerative and KMeans clustering while using cosine similarity to compute distances between sentence embeddings.The latter also uses cosine similarity for comparison between arguments however it adopts a greedy approach which reduces the list of arguments as clusters are identified. Experiments showed the performance of these two algorithms was very similar so further testing is required to more closely evaluate the differences between them.

I conducted an experiment comparing all clustering methods covered in this project; KMeans, Agglomerative, HDBSCAN, Community Detection, Hierarchical KMeans and Greedy Nearest Neighbour. Figure 4.1 shows the comparison of the cluster quality metric scorings for all algorithms with the bars in the graph showing the average precision, recall and F1 scores achieved across all topics in the data set and includes error bars representing the minimum and maximum scores achieved by a given topic. For each clustering method in this graph, the optimal parameters obtained in their respective experiments were used to show the best result achievable by the algorithm. For KMeans, this was a value of 4 for K as identified in table 3.2, a threshold value of 17.5 for Agglomerative clustering, as seen in figure 3.3. For HDBSCAN, a minimum cluster size of 3 was used for the final testing as while a value of 8 was found to achieve the best results in table 3.3, it did not perform as well when testing on the other topics. A threshold value of 0.825 and a minimum community size of 2 was used with the Community Detection algorithm as this was found to achieve the results as per figure 3.6. A minimum similarity threshold of 0.75 and a maximum threshold of 0.85 was used with Hierarchical KMeans as this was found to achieve the best results across all topics in figure 3.7. A threshold of 0.81 was used with Greedy Nearest Neighbour as it achieves the highest accuracy in figure 3.9.

Community Detection found clusters with the highest precision followed closely by Greedy Nearest Neighbour and then Hierarchical KMeans, while the other methods had comparatively lower scores for this metric. One of the most important requirements of this project is that the summary produced is accurate and representative of the arguments in the data set and this is why there is such a strong preference for a high precision score over recall. While it is simple to compare the precision scores between the algorithms, comparing the recall is less straightforward as some algorithms omit arguments and the figure does not show the size of the clusters returned by each method. Agglomerative clustering had by far the highest recall scoring, averaging above 0.9, however the precision was too low to be considered as a viable option. The reason behind this high recall value was that the algorithm would return a very low number of clusters, usually 2 or 3, and hence the recall was near to 1. This is one of the issues with this figure; it doesn't illustrate how well key points have been identified from the data set and this is far more important than the cluster scorings themselves.

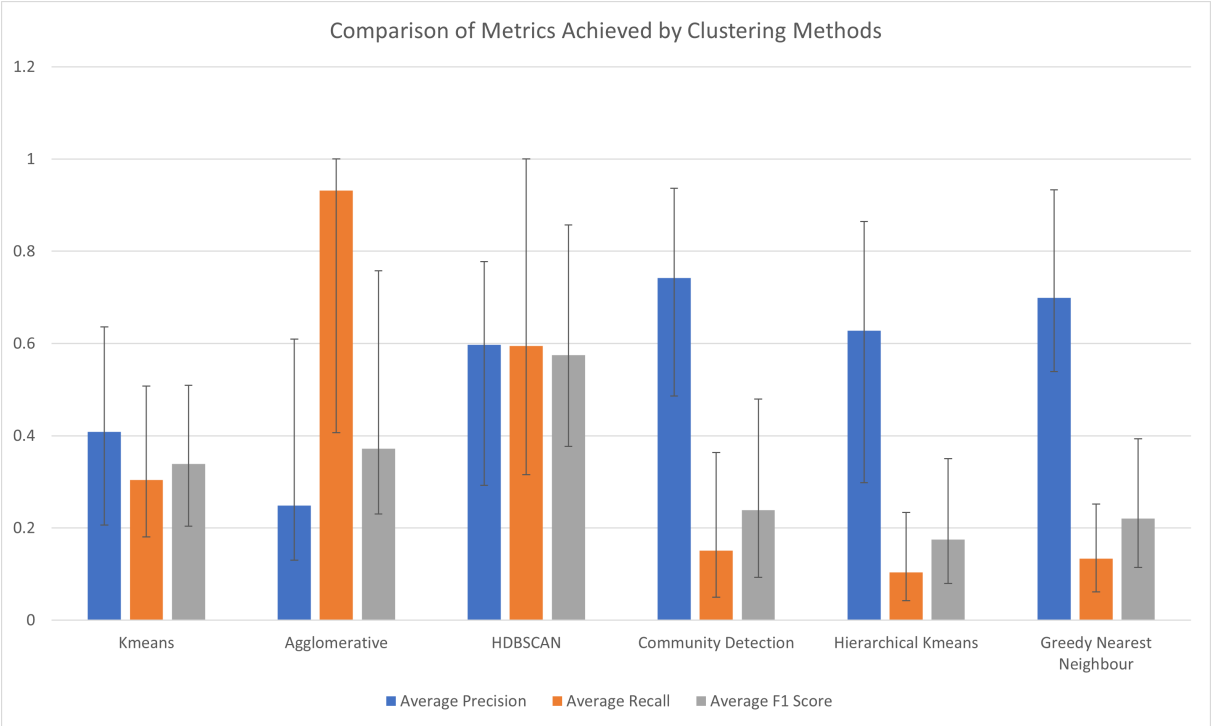

Figure 4.1: Comparison of metrics achieved by all clustering methods covered in this project

Figure 4.2 compares the difference in the average number of key points identified by each clustering method across all topics in the data set. The bars in the graph show the average percentage of key points identified across all topics in the data set and the average ratio between the number of clusters returned and the number of key points in that topic. A perfect algorithm would achieve 100% of key points identified and 100% of clusters returned. KMeans, Agglomerative and HDBSCAN all return too few clusters as it is below 100% for them all and therefore it is not surprising that they struggle to identify many key points. Hierarchical KMeans and Greedy Nearest Neighbour both identify 77% of key points in the data set, comparing this to Community Detection which identifies only 70%. The high number of key points identified comes at the cost of all three of these algorithms returning a greater number of clusters; 221% for Hierarchical KMeans, 205% for Community Detection and 200% for Greedy Nearest Neighbour. I think that this is a worthwhile trade off because it maximises the number of key points identified from the data set and while the final summary may have points which repeats themselves and a duplicates, it worth this for the increased accuracy.

With the results of this figure, I can conclude that the Greedy Nearest Neighbour algorithm performs best for the data set used in this project and therefore will be used as part of the final system. This final result means that on average approximately three quarters of key points are identified in each topic in the data set. The density of key points in the data set has a huge variance, where some key points may have over a third of the topic's arguments assigned to them while smaller ones may only have 3 or 4. This may be the reason some key points are missed as they get clustered along with larger key points or simply discarded as outliers or noise. However, this can be seen as a good thing if we apply this to the context of misinformation or unfounded arguments from questionable sources. For example, if this system received a list of arguments from the internet as input and had to summarise them, uncommon arguments which aren't made very often would be seen as outliers and in most cases not included in the final summary. The main issue with the algorithm I have developed is the number of clusters returned is on average over double the key points in the data set. These key points were written by an expert debater and therefore the conclusion can be made that while these algorithms cannot compete with the conciseness of a human manually writing key points, it is substantially faster.

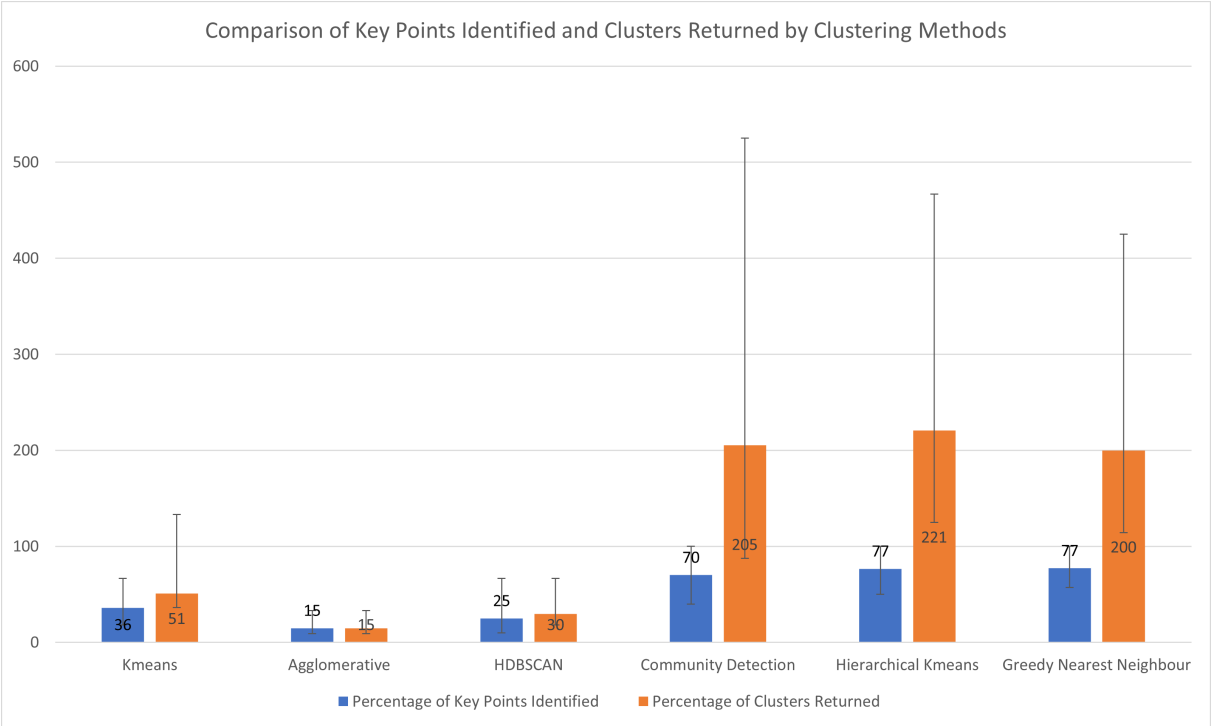

Figure 4.2: Comparison of the number of key points identified and the number of clusters returned by all clustering methods covered in this project

Experiments showed that summaries generated using the extractive Most Central Argument approach were of the highest quality compared to summaries generated with other methods (see table 3.8). For example, the abstractive Pegasus model suffered from hallucination and would include inaccuracies in the summaries generated and as such could not compare to the quality the extractive method offered. Considering this, the Most Central Argument method was selected to form part of the final system.

To demonstrate the performance of the final system, I tested it on all topics in the data set and I will present the best and worst resulting summaries generated to provide a representative showcase of performance. Table 4.1 shows the summary generated for the arguments in the topic of "We should subsidize vocational education" where 6 out of 8 key points were identified and a total of 12 generated key points. Table 4.2 show the summary generated for the arguments in the topic of "We should fight urbanization" where 4 out of 7 key points are identified and a total of 19 generated key points. These results suggest that this system may need require adjustment before it can be used outside of this project.

Table 4.1: Final Summary result for the highest rated topic of "We should subsidize vocational education"

ID	Data Set Key Point	Generated Key Point
0	Subsidizing vocational education leads to an influx in those professions	
1	Vocational education is not a good career choice	We shouldn't subsidize vocational education because jobs in that field are often low paying
2	Subsidizing vocational education diverts money from more worthy causes	Subsidizing vocational education takes money away from other, more worthy causes that help society at large.
3	Subsidizing vocational education is expensive	
4	Subsidizing vocational education promotes those with fewer resources	Vocational education should be subsidized as the majority of those studying would be from lower income families. We should subsidize vocational education because it would give more people a chance to go to college that can't afford it.
5	Vocational education is a good career choice	Subsidizing vocational education would prepare people for careers and jobs.
6	Vocational education better fits many students	Vocational education allows students who aren't suited to traditional college educations to learn skills needed for the workforce
7	Vocational education is beneficial for the entire market/society	We should subsidize vocational education because there is a shortage of tradesmen and women. Vocational education allows to develop citizens with skills still necessary for our society. Vocational education provides the training for valuable skills - money spent that will be return many times over in future economic prosperity. All education including vocational education should be subsidized. it all leads to the bettering of society overall. Vocational education is key to getting more skilled workers. We should subsidize vocational education because everyone relies on skilled tradespeople.

Table 4.2: Final Summary result for the lowest rated topic of "We should fight urbanization"

ID	Data Set Key Point	Generated Key Point
0	Cities offer more opportunities	urbanization provides more available jobs for people we should not fight urbanization because it gives more people places to live urbanization allows easier access to essential services such as healthcare and education cities bring everything closer together and make it easier to do everything as a result of that. urbanization goes hand in hand with mass transit, leading to more jobs and an easier way to get to those jobs, which benefits everybody. urbanization is key to an efficient modern community. it ensures we have access to utilities, hospitals, food shops, water and for the lowest cost per person urbanization will increase available housing which we need.
1	Urbanization benefits the economy	urbanization is important for the growth of economies
2	Urbanization benefits the environment	
3	Restrictions on migration would benefit people in the rural areas economically/socially	
4	Urbanization causes crime	urbanization creates overcrowded places with high crime
5	Urbanization harms the environment	urbanization is causing more pollution and hurting the environment urbanization destroys natural habitats for animals and plants. we should fight urbanization because of the environmental impacts it has on the planet. urbanization can cause an increase in crime rates and pollution urbanization contributes to overcrowding, poverty and environmental degredation. urbanization has caused too much destruction of wildlife habitat and should be fought against. we are losing species left and right due to this. we need to have some untouched land for crops and nature urbanization destroys green space, which has been shown to have a detrimental effect on human mental states. urbanization is one of the causes of climate change we should fight urbanization because it causes smog and air pollution
6	Urbanization is causing a strain on the cities' resources	

Chapter 5

Conclusion

5.1 Contributions and Achievements

This project has developed a system for clustering similar arguments based on their semantic similarity and summarising them to return a list of the most important points made from them. In this paper, I have introduced two novel algorithms in this paper, Hierarchical KMeans and Greedy Nearest Neighbour. The former takes inspiration from and attempts to combine the benefits of Agglomerative and KMeans clustering and as such starts with every argument in their own cluster and iteratively merges clusters, based on a minimum and maximum similarity threshold, recalculating cluster centroids until it converges to a final state. The use of two threshold values allows for flexibility in the trade off between precision and recall of the clusters returned. This algorithm was able to identify on average just over 75% of the key points from the data set, key points which were written by an expert debater. This algorithm performed the very well and better than all standardized methods tested; KMeans, Agglomerative, HDBSCAN and Community Detection and was only out performed marginally by the Greedy Nearest Neighbour algorithm. This algorithm takes a greedy approach to clustering which uses cosine similarity to identify lists of similar arguments and creating clusters until there are no more lists of similar arguments of size 2 or greater. This algorithm offers slightly better performance compared to Hierarchical KMeans by on average returning 9.5% fewer clusters. Both of these algorithms could be used in many other applications and need not be confined to only clustering sentence embeddings but any data with a similarity matrix.

5.2 Project Status

All technical objectives were met. That is to say:

- I conducted research into NLP models and found SBERT is the best for computing semantic similarity between sentences by considering the context in which words are used.
- Hierarchical KMeans clustering was developed and performs better than all other existing algorithms tested.
- Greedy Nearest Neighbour clustering was developed and offered marginal improvements over Hierarchical KMeans
- Extractive and Abstractive text summarization methods were compared, concluding that abstractive summarization is too prone to errors and not reliable enough to use.

5.3 Future Work

The one problem with both algorithms is that they return too many clusters, on average double the number as key points in the data set. This therefore leads to the final list of key points generated, the summary, being longer than necessary and containing a few quite similar sounding points. I believe that the limiting factor is the SBERT model which generates the embeddings for each sentence and argument. The distance between arguments assigned to different key points needs to be far greater to allow for better clustering however this is difficult because while the arguments have differing main points, they are still related to the same topic and therefore will maintain a large degree of similarity. This is why

there is such a trade off between precision and recall as the variance in the computed similarities between arguments is not large enough. In this project, precision was prioritised over recall because I think it is more important to capture a broader range of arguments and viewpoints in the data set than attempting to reduce repetition of similar key points in the final summary. This is the justification of the final system returning on average 2 clusters per key point in the data set as this is the expense that comes with the increased number of key points identified.

To improve my result, I would need to improve the NLP model so that it generates embeddings which could be clustered more easily. While I tried to fine tune the model at the start of the project, I was not successful in improving the scores for both metrics it was tested against, intra-cluster and inter-cluster similarity. The fine tuned model was better able to identify similar arguments, with an increased similarity scores between arguments however, the similarity scores between dissimilar arguments also increased, counteracting any previous benefit and actually causing the model to on average perform worse than before it was fine tuned. I believe that using more training data may have alleviated this issue and allowed the model to improve at distinguishing between arguments. Another reason for the algorithms performance is that there are on average over 150 arguments per topic in the data set and hence the embeddings are destined to be quite similar if the topic of them is the same. This is something that cannot be controlled without changing the data set however even then, the purpose of the project is to combine arguments about the same topic. Another issue is the variance in key point distribution in the data set, with some key points accounting for more than 30% of the arguments in some topics. Again, this cannot be controlled and may actually be a good illustration of arguments which would be obtained from the internet.

An extension to this project could be to further develop the system so that arguments are automatically obtained from online sources using argument mining techniques and this system used to generate summaries from them. This could be developed into a search engine type website where the user could search for a particular topic and be presented with a list of important arguments for and against, with a list of sources included. I believe this would be very useful and would allow people to inform themselves on issues quickly, easily and accurately. However, for this, there would need to be run time improvements to all stages in the system; generating the sentence embeddings, clustering them and generating summaries. One way is to attempt to speed up the sentence embedding model and this would form an interesting area for further study, attempting to speed up the neural network behind it. It would be interesting to try using multiple sentence embedding models in a concurrent system to utilise multiple cores of possibly a server where the website is hosted. A way to speed up the clustering could be to apply a dimensionality reduction on the sentence embeddings like UMAP to reduce the computation required to calculate similarity between two embeddings in the vector space. It would be interesting to see the effect this has on both the run time of the clustering algorithm as well as the accuracy of it.

I theorise that some models are better at classifying similar arguments while others are better at the opposite, classifying dissimilar arguments and by combining the use of both models in tandem could lead to a boost in performance. This could come in the form of an increase in the number of key points being identified or a reduction in the clusters returned by the algorithm or potentially both. More generally, this project could form the basis of further research and development into the task of automatic argument summarization.

Bibliography

[1] Roy Bar-Haim, Lilach Eden, Roni Friedman, Yoav Kantor, Dan Lahav, and Noam Slonim. From arguments to key points: Towards automatic argument summarization, July 2020.

[2] Roy Bar-Haim, Yoav Kantor, Lilach Eden, Roni Friedman, Dan Lahav, and Noam Slonim. Quantitative argument summarization and beyond: Cross-domain key point analysis. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 39–49, Online, November 2020. Association for Computational Linguistics.

[3] Ricardo Campello, Davoud Moulavi, and Joerg Sander. Density-based clustering based on hierarchical density estimates, 04 2013.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[5] Ira Bruce Gaultney, Todd Sherron, and Carrie Boden. Political polarization, misinformation, and media literacy. *Journal of Media Literacy Education*, 14(1):59–81, 2022.

[6] Alison J. Head, John Wihbey, P. Takis Metaxas, Margy MacMillan, and Dan Cohen. How students engage with news: Five takeaways for educators, journalists, and librarians. executive summary. Publisher's version.

[7] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38, mar 2023.

[8] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents, 2014.

[9] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries, July 2004.

[10] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction, 2020.

[11] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[13] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks, 2019.

[14] Myrthe Reuver, Suzan Verberne, Roser Morante, and Antske Fokkens. Is stance detection topic-independent and cross-topic generalizable? – a reproduction study, 2021.

[15] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

[16] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization, 2019.

[17] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert, 2020.