Bachelor's Thesis Computing Science

Radboud University Nijmegen

Unsupervised Topic-Based Sentiment Analysis for Twitter

Leveraging Transformer Models for Measuring Public Opinions

Author:
Mick van Beijnen
s1055989

First supervisor/assessor:
prof. dr. ir. Arjen P. de Vries

Second assessor:
dr. ir. Faegheh Hasibi

June 28, 2023

Abstract

This thesis proposes an algorithm that utilizes Large Language Models (LLMs) for clustering social media posts based on topics and performing sentiment analysis. The study addresses the challenges of analysing vast and non-uniform social media data and highlights the benefits of LLMs in improving accuracy and efficiency. The methodology incorporates Transformer models, cluster analysis, dimensionality reduction techniques, and machine learning models for sentiment analysis.

For displaying the effectiveness of the algorithm, we apply it on a dataset of Tweets about ChatGPT. The results demonstrate the algorithm's ability to correctly capture topic clusters and sentiment distributions within these topics. The findings contribute to social media analysis and offer insights into public sentiment.

Contents

Chapter 1

Introduction

1.1 Background

Social media allows us to discuss topics with much larger groups of people, faster than ever before. Platforms like Twitter, Facebook, or Reddit allow users to express their opinions and then other users from all around the globe are able to read these statements and respond to them with just the click of a button. Depending on the platform used, these posts can be either the length of a single sentence, or the length of an entire essay, giving us both in depth looks into issues, or short and comprehensive insights into the minds of others.

The ease of use for these platforms, however, can act as a double-edged sword. While more perspectives being shared than ever before is good for healthy discussions, the large amount of posts can make it difficult to find specific information, or analyse all the different perspectives to reach a consensus. Especially since users are given a lot of freedom in how they format their posts. The result of this is that anyone interested in analysing this online discourse is going to have a hard time working through these huge amounts of non-uniform data.

Having a proper method of studying these posts could thus prove to be useful to anyone interested in studying what is being said on social media. This is not only of interest to social scientists, but also useful for companies for example, as knowing the general public's stances on certain issues will aid in making better business decisions. A lot of prior info on online reputation monitoring already exists, as can be seen in the work by Carrillo-de-Albornoz et al. [5], but there is still plenty of room for improvement. For example, the methods used in that paper rely on supervised machine learning methods and therefore depend on annotated training and test data.

Now with the recent breakthroughs made in large language models (also called LLMs for short), it may be possible to develop an unsupervised, or semi-supervised alternative method. Using this new technology we might

decrease the computing time, improve on the accuracy of the algorithm and/or reduce the level of supervision required.

1.2 Overview

This gives us the following research question:
How effective are LLMs in clustering social media posts by topic and analysing sentiment?

We will create one such algorithm, by using Large Language Models both for the filtering and grouping of posts, and the sentiment analysis of the contents of these posts. This process is easily automatable and will give us the proper data to see whether the public has mostly positive or negative feelings towards a given subject.

We evaluate our approach using a case-study methodology, specifically focusing on the analysis of opinions regarding OpenAI's ChatGPT found on Twitter. This will be done by first providing the required background information. Secondly, all the steps taken in order to create this algorithm will be explained, as well as the method in which its performance is evaluated. Thirdly, the results of the study will be presented Fourthly, the results of the study will be discussed. Finally, the conclusions reached in this paper will be laid out.

Chapter 2

Preliminaries

2.1 Transformer Models

To better understand our approach for clustering topics and performing sentiment analysis, it is helpful to have a basic understanding of the Transformer models that will be used. These models are a recent advancement in natural language processing (NLP) first introduced in a 2017 paper by Vaswani et al. titled "Attention is All You Need" [21], and have been particularly effective at processing large amounts of text data.

2.1.1 How Transformers Models Work

Transformers are composed of an encoder and a decoder, each of which consists of multiple layers of self-attention and feedforward neural networks. The encoder is responsible for processing the input sequence, while the decoder is used for generating output sequences, such as in language translation tasks.

The key innovation of transformers lies in the use of the self-attention mechanisms to process the input data. Self-attention allows the network to selectively attend to different parts of the input sequence, based on their relevance to the current task. This makes transformers particularly effective at processing long sequences of text, which can be difficult for traditional neural network architectures to handle.

The self-attention mechanism in transformers uses an attention function that maps a query and a set of key-value pairs to an output. In this context, the query, key, and value vectors represent the input sequence's tokens (words or subwords) in three different projections. The query vector represents the current input token being attended to, while the key and value vectors represent all the other tokens.

The attention weights are computed based on learned query, key, and value vectors that are multiplied to compute a scalar attention score for each position in the input sequence. These attention weights determine how much

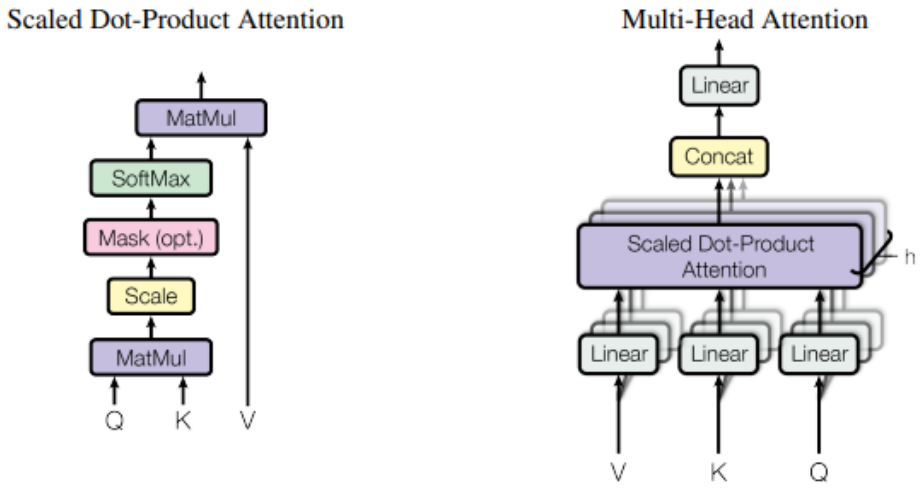

Figure 2.1: (left) Scalar Attention Score (right) Multiple Attention Layers Running in Parallel

each token contributes to the output for the current query. This process is done for each layer of the network, allowing the network to attend to different parts of the input sequence at different levels of abstraction. These different layers are all calculated in parallel, in order to keep the algorithm efficient. A visual explanation of this mechanism can be seen in Figure 2.1, taken from the work by Vaswani et al. [21].

2.1.2 The Benefits of Transformer Models

One of the key advantages of using self-attention is that transformers are now able to process input sequences in parallel, rather than sequentially like other neural network approaches. This makes them much more efficient for processing long sequences of text, and has led to state-of-the-art performance on a wide range of natural language processing tasks.

Another benefit of the transformer models is that they are also remarkably user-friendly. As can be seen in the DIXIT article "Het Succes van BERT en Huggingface - Taaltechnologie in 5 Regels Code" by Suzan Verberne, implementing basic functionalities like Sentiment Analysis using transformer models [22] only takes a few lines of code.

2.2 Cluster Analysis

Cluster analysis is a data mining technique that divides data into groups (clusters) that are meaningful, useful, or both [20]. These clusters are able

to capture the natural structure of the data, making the data easier to understand while preserving the characteristics of the data, or serve as a useful starting point for further processing.

2.2.1 Similarity Measures

In order for an algorithm to consider which data points to group together, it needs to be able to calculate how similar or dissimilar two data points are. To accomplish this, a similarity measure (sometimes called distance metric) is used. There are several similarity measures to choose from, but the most frequently used are:

- Euclidean Distance: Measure of similarity as the straight-line distance between two points in space. It considers the coordinates of the points in all dimensions and provides a direct measurement of how far apart they are.
- Manhattan Distance: Measure of the distance between two points by summing the absolute differences of their coordinates along each dimension. It reflects the total number of blocks one must travel horizontally and vertically to reach from one point to another in a grid-like path.
- Cosine Similarity: Measure of the similarity between two vectors based on the cosine of the angle between them. It is used to measure the direction and magnitude of similarity, particularly in high-dimensional spaces. Cosine similarity is unique in that it ranges between -1 and 1, with a higher cosine similarity indicating a higher degree of similarity between the vectors.

2.2.2 How Cluster Analysis Works

The main job of cluster analysis is to group data points together that share similarities in some characteristics, while being dissimilar to data points in other clusters. The way this is done can vary greatly, as there exist many algorithms for clustering that all work differently. The general process, however, does remain the same over the different algorithms.

1. First, the data needs to be represented in a suitable format to apply clustering analysis. Most often, this will be a numerical representation.
2. Once the data is in a usable format, a similarity measure needs to be chosen, so the algorithm is able to calculate how similar two data points are. This determines whether to cluster these points together based on the result.

3. Now that we have chosen how to define similarity, we can initialize our clusters. Depending on the algorithm, the clusters can be either initialized randomly, or by using a predetermined number of clusters.

4. After having created our initial clusters, we start iteratively updating them by assigning the data points to their best fitting cluster. This depends on what clustering algorithm and what similarity measure is chosen, the way this happens can vary.

5. Finally, the algorithm converges or reaches a stopping criterion. At this point we have our final clusters. Now some evaluation metrics might be applied over the final clusters in order to gain confidence in the performance of the clustering algorithm. If satisfied with the results, the data is ready to be interpreted or processed further.

2.2.3 The HDBSCAN algorithm

For the purposes of this paper, we will be using the HDBSCAN algorithm, which is a current state of the art clustering algorithm. HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering algorithm first introduced by Campello et al. in 2013 [3]. It is an extension of the DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm, that adds a hierarchical approach and a robust method to determine cluster density thresholds [7].

HDBSCAN works by performing the following steps:

1. First, HDBSCAN identifies areas of high density in the data by assigning each data point a reachable distance, which is the maximum distance within a data point must be to be considered in the same cluster, and a minimum cluster size. It then identifies "core points", which are data points with many neighbours. These points will be considered the centres of the clusters.

2. After having identified the core points, a MST (Minimum Spanning Tree) will be constructed between all the core points based on the distances between them. This MST represents the hierarchical structure of the data. The root of the tree is the core point with the highest density, while the leaf nodes are the low density points.

3. This hierarchy is then used to merge clusters together. The highest density clusters will first be merged together, and the algorithm then moves on to the clusters with lower density levels. This merging continues until either the desired number of clusters is reached, or a stopping criterion is met.

4. Finally, once the algorithm has stabilized, it will detect outliers based on their distances to the clusters. This way, noise will get removed from the data, resulting in cleaner clusters.

2.2.4 Sentence Embeddings

Since HDBSCAN requires the input data to be in a numerical format, we will be using sentence embeddings to represent our text data. While the concept of embeddings exists for quite a while, they were popularized in Church's 2013 paper Word2Vec [4].

While representing words as numbers is not new, embeddings are special in that they are able to retain their syntactical and semantic meaning from their original context. They are able to do this differs per algorithm. In the case of Word2Vec, unsupervised learning models were used to train the model. These models consider the distributional properties and co-occurrence patterns of words in a given corpus to learn meaningful word representations. A limitation of this method, however, is that the vector representations for words are fixed based on what was learned during the training of the model. It thus is not able to assign different values to words used in different contexts, but rather assigns a general value it learned.

More recent models fix this problem by making use of the self-attention and feedforward neural networks of transformer models. Using these to generate the embeddings makes it possible to process the input tokens bidirectionally, so both the context to the right and left of a word is being preserved. They then output high-dimensional vectors of a fixed size, each representing a token and its context.

2.2.5 Curse of Dimensionality

Applying clustering analysis on data with a high dimensionality can cause some problems. In a high dimensional space, data can be very sparse and contain a lot of irrelevant features or noise. This makes it harder to correctly group similar data together. Furthermore, some algorithms are not able to handle data with high-dimensionality, and will take a much longer time to run if it has to compute a lot of more data. To solve these issues, dimensionality reduction techniques exist [20].

Dimensionality reduction algorithms transform the data from its high-dimensional space to a lower-dimensional space, while preserving the structure and important characteristics of the original data. The goal is to find a new, smaller set of variables, known as derived variables or features, that capture the essence of the original data. Using this new, smaller set of features will help speed up the algorithm, as well as make the data easier to interpret for us.

2.3 Sentiment Analysis

Sentiment analysis, also known as opinion mining, is a process that involves determining the subjective nature (sentiment) of a piece of text, first introduced by Pang et al. in their 2002 paper "Thumbs up? Sentiment Classification using Machine Learning Techniques" [15]. The algorithm aims to identify and extract the underlying sentiment expressed by the author, whether it is positive, negative, or neutral.

Sentiment analysis algorithms typically perform the following steps:

1. First the text is preprocessed by removing irrelevant information like punctuation, stop words, and special characters. In the case a transformer model is used (like we will be doing) this step also includes transforming the text into tokens.

2. Next, the relevant features are extracted from the preprocessed text. These features are the parts of speech, sentiment lexicons, or word embeddings that capture the semantic meaning of words.

3. Once the features are extracted, the model is able to assign a sentiment label to the text. The sentiment label could either be a binary classification (positive or negative) or a multi-class classification (positive, negative, or neutral).

These algorithms rely on well-trained machine learning models in order to correctly classify the text. The models learn patterns and relationships between extracted features and their corresponding sentiment in a labelled training set. After being properly trained, a model will be able to correctly classify new data. These days, many pre-trained models exist and are easily available to use either straight out of the box, or to be tweaked to suit a specific use case.

Chapter 3

Methodology

3.1 Clustering using BERTopic

The first step of the algorithm will be the clustering the documents into different topics. To do this, the BERTopic topic modelling algorithm created by Maarten Grootendorst is used [9]. BERTopic clusters text data (and since recently also images) based on the topic of the contents, precisely what we want to do with our documents.

3.1.1 The BERTopic Model

BERTopic is a modular approach that consists of in 6 main steps. This means that we are free to pick and choose algorithms for each step, or even leave some out entirely, to best fit our scenario. Aside from being highly customizable to our specific needs, it also means the method is future-proof. When new, more accurate or faster algorithms are developed for any of these 6 main steps, they can easily be integrated into the algorithm, making the overall process of clustering the data more accurate and faster.

The six main steps performed by BERTopic are:

1. Embedding documents: convert input documents into numerical representations called embeddings. These embeddings capture the semantic meaning of the documents inside a high dimension vector. The different dimensions of the vectors will allow the clustering algorithm to group similar data.

2. Dimensionality reduction: reduce the dimensionality of the document embeddings using dimensionality reduction algorithms. This step aims to preserve the local structure of the high-dimensional data in a smaller, more comprehensive data set. The reduced dimensionality of the data simplifies data interpretation and enhances the effectiveness of the clustering algorithm.

3. Clustering: The reduced-dimensional embeddings are then clustered using a clustering algorithm that groups similar documents together. These algorithms identify patterns within the data and group documents based on their similarities.

4. Tokenization of topics: After the documents are clustered, break down the topic representations into individual words or phrases. This step allows for a more granular analysis of the topics.

5. Weighting tokens: assign weights to the tokens based on their importance within each topic. This weighting process helps highlight the most relevant and informative keywords within each topic. For token weighting, c-TF-IDF (class-based Term Frequency-Inverse Document Frequency) or one of its variants are used. This formula considers the importance of terms within each cluster based on their frequency and distribution across the entire document collection.

6. Topic representation: generate topic representations by selecting the most important terms for each cluster based on their weighted values. These terms reflect the key concepts and themes present in the documents within each cluster, and the top terms can thus be used as a representation for the overall topic of the cluster.

The exact implementation for these 6 steps is yet to be defined. In the following sections, the methods used in this thesis will be explained, as well as the motivation behind doing so discussed.

3.1.2 BERT Embeddings

For the first step of transforming the text data into a numerical format, we will be using BERT embeddings. The BERT (Bidirectional Encoder Representations from Transformers) model is a pre-trained model first introduced in BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding by Devlin et al. [6]. This model produces BERT embeddings, which are numerical vector representations of tokens (words or subwords), that retain semantic and syntactic information of words in their context.

These BERT embeddings are implemented by using the SentenceTransformers model, a Python framework for state-of-the-art sentence, text and image embeddings, created by Reimers et al. [18]. Since we want to design an unsupervised approach, we will be using a pre-trained model. Specifically, we will be using the all-MiniLM-L6-v2 model by Reimers et al. [14], as this model is 5 times faster than the most accurate all-mpnet-base-v2 model, while still offering good quality [17]. Thus, this model is a computationally efficient model, that also gives us reliable embeddings.

3.1.3 Dimensionality Reduction using UMAP

After the first step, the data will be formatted correctly, but still represented in high-dimensional vectors. Clustering high dimensional data is a challenging problem, so we will first reduce the dimension of the embeddings obtained in the previous step. This is because in a high dimensionality, the data can be very sparse and contain a lot of irrelevant features or noise. Reducing the number of dimensions thus can help clean up the data, which will lead to better performance. There are many dimensionality reduction algorithms to chose from, but in this paper the state of the art UMAP algorithm is used.

UMAP (Uniform Manifold Approximation and Projection) is a dimensionality reduction technique first introduced by McInnes et al. [13]. UMAP works by performing the following steps:

1. First, the nearest neighbours for all data points are identified based on a chosen similarity measure. A weighted graph is then constructed, where each point is connected to its neighbours with the weight of their similarity.

2. We then initialize a lower dimensionality version of the graph. All data points are plotted in this new graph in such a way that neighbouring data points retain their similarity score.

3. We now iteratively update the graph by comparing the new graph to the old graph. A data point is selected and gets moved closer to its nearest neighbour, and further from a non-neighbouring point in the low dimension space by comparing it to the original graph. This happens until a certain stopping criterion is met.

Depending on the clustering algorithm selected, we can adjust the target dimensionality. In our case, we choose a clustering algorithm that still works well with 5–10 dimensions. Since the lower the number of attributes we reduce to, the higher the chance of losing important information, it is wise to leave the data with a bit more variables. To control the target number of dimensions, we utilize the UMAP parameter `n_components`. For this algorithm, we set the value of this parameter to five. This value was chosen as a starting value after seeing multiple example implementations of BERTopic use it [8][10]. However, after some further experimentation with different values we decided to leave it at five, as it seemed to reduce the dimensionality enough so that sparseness and noise in the data is reduced and the clustering efficiency improves, but it is not so low that it led to a significant loss of the original data structure.

Because we will be working with high dimensional vector data, we make UMAP use the Cosine similarity measure. The Cosine similarity uses the angle between the two vectors to measure the direction and magnitude of

similarity. This angle based approach makes it well suited for application on high dimension data.

Other parameters worth specifying are the `n_neighbors` parameter, which determines the number of nearest neighbours considered for each data point during the construction of the neighbourhood graph. Having a higher number of nearest neighbours captures more fine-grained details of local data, but also leads to a higher computation time. We set the number of nearest neighbours to 10, as we are only interested in grouping documents that share the same topic, which does not require a high level of detail, preferring the efficiency gain over the loss of accuracy.

We also have the `min_dist` parameter, which controls the minimum distance between points in the low-dimensional embedding. Having a lower minimum distance between points results in more densely packed clusters, and a higher number in more spread out clusters. Whether you want more dense or more sparse clusters depends on the way the data is structured. After some trial and error, very densely packed clusters were found to give the best results, so the min_dist parameter is set to 0.0.

3.1.4 Clustering using HDBSCAN

The next step is to cluster the lower dimensional data, where we use the art HDBSCAN algorithm.

Just like how the UMAP algorithm had to be able to calculate how similar two data points are to each other in order to maintain the data structure in a lower dimension, HDBSCAN will need to be able to measure similarity in order to know which points to cluster together. However, HDBSCAN does not support the cosine similarity metric. That is why the Euclidean distance metric will be employed in this instance, which calculates the distance of two data points simply by taking the straight line distance between them.

The next decision we have to make is how sensitive we want the clustering to be. To control this, we can set the `min_samples` parameter, which determines the minimum number of samples in a neighbourhood for a point to be considered a core point, to a smaller number. With a lower value the clustering will become more sensitive, while with a higher value data points would be labelled as noise much earlier. We wanted as many documents to be clustered as possible, so we set the value for the minimum samples to the low value of 10.

An additional reason why we are able to leave the minimum number of samples so low is because we use the EOM method (Excess of Mass) cluster selection method. This approach works by looking at the size and density of clusters and picks the clusters (clusters with a large excess of mass) that are the largest and densest compared to their parent cluster. Since EOM considers density in its calculation, just like HDBSCAN, they

work well together and because prominent and dense clusters take priority, the algorithm will be effective at handling noise and outliers.

Another parameter that is worth experimenting with is the `min_cluster_size` parameter, which sets the minimum number of data points required to form a cluster. In this instance, this value is set to 150. Having a higher number here comes with two benefits. First, the higher the minimum number of data points, the shorter the algorithm will take to run. This is because instead of many small clusters, we will get less big clusters.

Second, a small cluster is not valuable to us. We still want to apply sentiment analysis later on, and for this to run reliably, we want a big enough sample size for each cluster. Since we will also be filtering out documents of which the clustering algorithm is unsure whether it belongs to the topic, and documents of which the sentiment analysis is unsure whether it correctly identified the sentiment, we know the size of the clusters is likely to decrease later on. Having the initial cluster size thus be quite large, gives us some leeway to lose a few samples along the way.

That being said, raising these numbers too high will lead to clusters that are so broad in topic, that we will not be able to obtain a detailed sentiment analysis. The whole step of clustering would become trivial when this happens. That is why we decided to still keep the number at 150. We wanted the sentiment analysis to applied to clusters of at least 100 samples, to still get reliable results. Setting the minimum cluster size to 150 gives us enough leeway that we can be confident that more than 100 samples will be left after the filtering of the data, while also leaving enough room for HDBSCAN to be flexible in the clusters it creates.

Finally, we set the `prediction_data` flag to be true. This allows HDBSCAN to cluster new, unseen data points later on. While this is not useful during the initial clustering, being able to assign new data to existing clusters can become useful later on. Several techniques will become available such as the real-time clustering of separate data sets, monitoring data drift to detect changes over time, incremental learning and transfer learning. It also allows for adding more data to the clusters after the initial clustering. Having these extra options comes at the price of the clustering taking a bit longer, but the added flexibility we gain to do more operations on the data later on outweighs this negative.

3.1.5 Tokenization of Topics using CountVectorizer

For the tokenization of topics, the Scikit-learn class CountVectorizer[16] will be used. This class tokenizes the text, builds a lexicon of unique words or n-grams, and then creates a matrix where each row represents a document and each column represents the count of a specific word in that document. This word frequency matrix is what will be used to identify what words (topics) each cluster represent.

By default, CountVectorizer only tokenizes unigrams (single words). However, certain topics that we would like to include in our topic representation are made up of several words (programming assistant, for example, is two words). To make sure CountVectorizer is able to create these groups of words, we use the `ngram_range` parameter. This parameter takes a tuple of two numbers, of which the first represents the minimum number of words tokenized, and the latter represents the maximum number of words tokenized.

Ideally we would set the lower bound to one, and the upper bound to three so that the algorithm would be flexible in how it wants to define its topics. However, doing so would require more memory, as is this case the algorithm has to take all unigrams, and all possible combinations for bigrams and 3-grams into consideration. Due to the limited resources available for this study, this simply was not possible. That is why we set this parameter to only consider bigrams, which does allow for some extra context, while also being possible with the amount of memory available to us.

Depending on the application domain, specific preprocessing of the text may be desirable, such as the removal of stop words or domain specific terminology. We discuss this in section 3.3.2 about the case study.

3.1.6 Weighting Tokens & Topic Representation

The weighting of the tokens and the Topic Representation step allow us for extracting meaningful and coherent topics from a text corpus using BERTopic. However, due to the time constraints on this research, these steps of BERTopic could not be experimented with extensively. This, combined with the fact that the tokenization step already performs worse than ideal due to the memory constrains, we were unable to come to any meaningful conclusions for this part of the algorithm. For the purposes of this paper, the topic representations were used as they left the tokenization step for debugging purposes. In order to still neatly present the final data, new topic representations will be generated during the Sentiment Analysis part of the code in a simpler manner.

3.1.7 Creation of BERTopic Model

Having initialized all the models needed to perform the steps of BERTopic, we can now create the BERTopic model. We pass the UMAP, HDBSCAN and CountVectorizer models we initialized to the BERTopic model as parameters.

Alongside these parameters, we also specify a few others. The first of these is the `language` parameter. This parameter specifies the language to be used in the BERTopic model. It determines the underlying BERT language model that will be utilized, as these can be pre-trained on a specific

language, in order to increase performance. Since the value for this parameter will depend on the language of the specific data set that is being used, we will further discuss this parameter in section 3.3.2.

Parameter `top_n_words`, determines the number of words per topic that you want extracted. Too many words here, and the topics become less cohesive. Too few, and the topic clusters unrelated words. In our implementation, we set this value to be equal to 20. Since our list of stop words is not extensive, it is better to consider some more words in order to avoid most of the data being clustered together in just one cluster.

We can also choose how many topics we want to reduce to after training. If set to none, no reduction will be applied after training. If set to a specific amount, we will reduce to that specified number of topics. However, we opted for the third option, which is auto. In this option, BERTopic will use a heuristic to evaluate the current clustering, and the iteratively merges/splits clusters until the heuristic is optimized. A too large value will result in many clusters that share the same topic. A too low value, on the other hand, will cause different topics to merge into one cluster as well as causing the algorithm to take longer to run. Having BERTopic be in charge of deciding how many topics to reduce to will always result in a good clustering, regardless of the dataset that is used.

There is also the option to calculate the probabilities that each document belongs to the topic it is clustered to. We want documents with a low probability of belonging to the topic it is clustered to, to not be taken into consideration when doing the sentiment analysis. If the content of the document is actually about a different topic, including this document's sentiment into the analysis can interfere with the results of the overall topic's sentiment. That is why we take the extra time to calculate the probabilities, so that we are able to identify, and remove the documents with low probabilities.

Finally, since training the BERTopic model and extracting the topics can take a while, it can be nice to get some status updates along the way. We do this by setting the `verbose` flag to be true. The algorithm will now print a quick message whenever it completes one of its steps, with a timestamp. This allows us not only to keep track of how far along we are while it is running, but also to identify which steps are taking the longest. Having this knowledge can be helpful when tweaking the parameters for each model.

3.1.8 Training the BERTopic Model

Once the BERTopic model is created, the next step is to train the model and cluster the data. This is accomplished by utilizing the `fit_transform` function of the BERTopic model that was previously instantiated. The `fit_transform` function requires two parameters: the preprocessed documents and the embeddings model.

During the training process, BERTopic applies the embeddings model to the preprocessed documents, generating sentence embeddings. Subsequently, BERTopic employs the models provided as parameters in the BERTopic instantiation process, executing them in the correct order to create the clusters. This includes the dimensionality reduction using UMAP, clustering with HDBSCAN, and the tokenization of topics using CountVectorizer.

By performing these steps, BERTopic effectively trains the model and clusters the data, allowing for the identification of topics within the preprocessed document dataset.

3.1.9 Visualizing the Topic Clusters

Finally, after all the clustering is done, we visualize the clusters. This is not a necessary step for applying sentiment analysis, but it can give us insight into what topics exist in the data, and how prevalent they are. We visualize the clusters in two ways.

First, we use BERTopic's `visualize_topics` function. This plots all the topics on a 2D graph in circles, of which the size corresponds to the number of items in that cluster. This plot is mainly useful for checking which clusters are the big ones and which are the smaller ones. You might also see a lot of clusters close together.

While instinctively you might think these clusters thus share similar topics, you have to remember that we started off with very high dimensional data, of which we reduced the dimensionality and are now representing it in an even lower dimension. There may be some similarity between clusters that are near each other, but a lot of the structural information got lost by reducing the dimensionality so far, so this graph does not accurately represent how similar clusters are.

Second, we use BERTopic's `visualize_documents` function. This plots all the individual documents on a 2D plane, and colour codes them based on what cluster they are in. This makes it easy for us to visually identifying clusters, assessing their separation and overlap, detecting outliers, understanding cluster characteristics, recognizing patterns, and effectively communicating insights about the dataset.

3.2 Sentiment Analysis

Now that we are done clustering the documents into topics, we can move on to the second step of the algorithm: performing sentiment analysis. However, this is not as simple as just applying a sentiment analysis algorithm over the data we currently have. We first have to apply a bit of preprocessing again before we can use this algorithm.

3.2.1 Pre-Processing the Clustered Data

We are able to retrieve all the clustered documents into a pandas dataframe by calling the `get_document_info` function of the trained BERTopic model. This dataframe will be used as the starting point of the preprocessing process.

The first problem that needs solving is that not all documents were clustered with the same amount of confidence. The BERTopic model assigns a probability score to each document, indicating the likelihood of it belonging to the topic it got clustered into. This is because we set the `calculate_probabilities` flag to true in the creation of the BERTopic model. Documents that are placed in a cluster where they do not belong can interfere with the sentiment analysis process, since they are talking about a different topic on which the public opinion might be different. That is why we filter out all documents with a probability score below 0.5, so all the documents are more likely to be relevant to their assigned topics.

Next, this new dataframe with the filtered documents will be grouped based on their assigned topic. From this grouped data, we are able to generate a dictionary, which is a data type consisting of key-value pairs. In our case, the keys will be the topic numbers given by the trained BERTopic model, and the values will be a list containing all the documents clustered to the corresponding topic. Now, we can access all the documents of each individual topic.

Then we move on to the final part of the preprocessing process, which is the removal of unclassified documents. All the documents BERTopic is unable to find a cluster for, will be put together in one big cluster. Applying sentiment analysis on these documents will not give us any useful information about the public opinion on a certain topic, as the subjects of the documents can vary widely. BERTopic always assigns the topic number -1 to this cluster of unclassified documents, so we are able to remove all the unclassified data by checking whether the key -1 exists in the dictionary we created, and deleting it if it does.

3.2.2 Sentiment Analysis

Now, it is time to define the sentiment analysis function. This function will combine the process of analysing the sentiments, with some post-processing, and finally visualization of the data. Doing this all in 1 function, will allow us to easily call this process for all the different topics.

The TimeLMs paper by Loureiro, et al. [12] proves that the results of a particular analysis are better if the model is trained on documents from the same time period as the test data. Because the time frame of interest will of course vary for each application of this algorithm, we have to be able to swap in these models easily. To enable this, we use a pipeline approach, where the

model in question can be easily replaced as long as it outputs the sentiment in the same format. This ensures that the algorithm will be applicable to many research topics, while also maintaining a level of adaptability for future developments, similar to the approach used in BERTopic. If in the future we are able to train more accurate sentiment analysis models, those can also be used.

Since the format of the output is important for the rest of the code to work, we will describe the format here. Our model outputs a list of key-value pairs. The 'label' key represents the sentiment or classification label, and the 'score' key represents the confidence or probability associated with the assigned label. The label can be either 'positive', 'negative', or 'neutral' and the score is a value between 0 and 1. If another model is uses a different scale for the scores, or labels the sentiment using caps, or numbers, then some extra processing will be necessary to format them properly. Further discussion on the sentiment model can be found in section 3.3.2.

Post-Processing the Sentiments

After the model has generated the sentiments, we need to perform some post-processing before we are able to neatly present the results. First of all, we map each document to the sentiment it got assigned and the confidence score it got assigned with. Then, we proceed to remove all the documents with a low confidence score. The threshold for considered being low here was any score smaller than 0.6. Nearly all the confidence scores we are above 0.5, so we decided to be a bit harsher in removing any possible mistakes.

Now that we removed the documents that were not confidentially classified, we can count how often each sentiment appeared within the topic. With this information, we will be able to compare how often each stance on the issue appears, and thus find out whether a topic is controversial (all sentiments are equally represented) or if the public generally shares the same viewpoints (one sentiment appears far more often than the others).

Visualizing the Sentiments

Finally, we are ready to move on to the output of our algorithm. First, we are going to have to generate titles for the topics, and then we will create plots that visualise all the results.

Retrieving Key-Phrases

Because the time constraints on this research prevented us from fully implemented the Weighting Tokens & Topic Representation steps of BERTopic, we were unable to extract meaningful and coherent topics from a text corpus. However, analysing the sentiment becomes useless if we do not know

the subject that is being talked about. We thus have to find another way of retrieving the topics of each cluster.

For this, we use a slightly modified version of a code piece written by Matthew Schwarz [19]. We save this code as a function, so we are able to call it later on. This python script is fairly simple, and while there do exist many alternatives that would be more customizable to suit our needs like Rake, YAKE!, or TF-IDF for example, setting up these more complex algorithms properly would again take a lot more time to research and experiment with, which we simply do not have for the scope of this research. Furthermore, if we were to set up these more complex algorithms, we might as well fully integrate them into BERTopic, rather than use them separately.

This function works by looping over each document in the topic given as a parameter. Each document will get deconstructed into individual words, and then for all phrases, we keep track of how often it appeared. Phrases here are groups of words ranging in length from 1 word, to a chosen number of maximum length. Here we chose the maximum length to be three words, just like we initially wanted to do in the Tokenization of Topics step of BERTopic, but there we were limited by memory.

Since the phrases with length of one word will of course appear more often than phrases of length two or three, this algorithm implements a way of removing unnecessary sub-phrases. We do not want to just remove all small phrases, as some sub-phrases may already give enough information on their own. For example, Google as one word is a valid topic, but search engine we would rather have grouped together.

After having counted the number of occurrences for all phrases, the algorithm will filter out phrases that appear only once in order to speed up the next step. Then the remaining phrases are sorted based on how long they are, and we then loop over this sorted list to find sub-phrases. Only if the number of occurrences of the parent-phrase falls within a certain threshold of the number of occurrences of the sub-phrase, we remove the sub-phrase. The threshold here being set so that the parent-phrase's number of occurrences must be 75% or more of the amount of times the sub-phrase appears.

We made minor changes to the code to remove the preprocessing of the text data, as this is already done earlier on in our approach, as well as using our own list of stop words (the same as the one used for the clustering step, and the one that is discussed in section 3.3.2).

Printing the Charts

Now that we have all the necessary information, we can output the results. We plot the number of occurrences of each sentiment in a pie chart. A pie chart enables us to easily discern the ratio of different sentiments in a quick glance. This helps us determine if a topic is controversial and, if not, provides insight into the general opinion.

We title the pie chart in the following format 'Sentiment Analysis of Cluster `cluster` about: `title_str`', where `cluster` is the number corresponding to the cluster, and `title_str` is the title generated through the key-phrase function, both as parameters to the function. This way, it is also clearly visible from the graph what topic the cluster is about, and what key value we can use if we want to dive further into the specific documents of this cluster.

To gain even further insight into what the topic is about, and what perspectives people have on it, we have implemented a feature that 10 documents (or less if there are not that many) from each sentiment were randomly chosen. By hovering the cursor over a sentiment, these randomly sampled documents will be displayed. This way, we are able to view, in more detail, what is actually being said about the topic, and look at the different perspectives given.

Alongside the pie chart, we also display the number of documents in the topic, and the number of documents we were able to confidentially classify. While not immediately useful when just looking at the opinion on a topic, these values may give further insight into the overall engagement and scope of the conversation surrounding the topic.

3.2.3 Running the Analysis

The reason we made the retrieval of key-phrases and the sentiment analysis their own functions is so that we can easily apply them to all topics without the code becoming so large that it becomes cumbersome. Right now, we just implement a simple loop that iterates over all topics.

In each iteration, the key-phrase function first gets called, to determine how often each phrase appears in this cluster. The result of this function is then sorted on the number of occurrences in descending order. We then take the first five phrases of this sorted list, and call the sentiment function with these five phrases as a parameter, alongside the documents of this topic and the topic number.

3.3 Case Study

Now we have successfully created an algorithm that identifies the topics of social media posts, groups posts about the same topic together, and then analyses the sentiment expressed within each topic. However, we need a method to verify whether this algorithm actually works. To do this, we will be conducting a case study. We will be analysing the sentiments people express towards ChatGPT on Twitter, to see whether our approach produces the results we expect it to.

3.3.1 The Data

The first step will be to collect data to work on. Since for the purposes of this paper we are interested in finding the public sentiment on ChatGPT, it is best to use a dataset of only social media posts about ChatGPT, rather than using a dataset that contains about many other topics as well. This way, the filtering of relevant posts does not have to be done by the algorithm as well, which already drastically reduces the runtime of the algorithm.

More specifically, we will be analysing Twitter posts regarding ChatGPT. Since Tweets have a strict character limit compared to other social media platforms, the size of each post will be a lot shorter, which will shorten the computation time. Furthermore, limiting ourselves to one form of social media tends to lead to a more accurate clustering/sentiment analysis, as we can use models trained specifically for this social media platform.

A dataset that suits our needs is the ChatGPT Tweets first month of launch dataset by Minh Pham. This dataset contains all Tweets about ChatGPT from 30/11/2022 to 31/12/2022 gathered using the following query: `(chatgpt OR ChatGPT) lang:en -is:retweet -is:reply`[1]. This dataset was used simply because at the time work on this algorithm started, there were no alternative datasets that contained Tweets specifically about ChatGPT.

The Relevant Data

Now that we have our dataset, we load the data into a Jupyter Notebook. Since both the process of identifying topics to cluster, and analysing the sentiment requires just the content of the Tweets, we only need to use the tweet column of the dataset. Only taking this one column reduces the size of the data we are working with, and thus saves us some memory usage, which will increase performance.

While some other columns may also seem of interest in gathering public opinion, like the country the Tweet came from to measure sentiment per geographic region or the time the Tweet was created to gather sentiment over time, these are not very useful within the context of this dataset. The vast majority of Tweets do not have a country linked to them and by the nature of this data being gathered in the first month of the ChatGPT launch, there is not a lot of time to work with yet. Other data like the amount of likes, retweets, or comments could also be used to make certain popular or controversial Tweets weigh more in the sentiment analysis, but this proved to be outside the scope of this paper.

[1]The dataset "Minh ChatGPT Tweets first month of launch" by Minh Pham, `https://www.kaggle.com/datasets/pcminh0505/chatgpt-twitter`

Filtering the Data

The data is now loaded in, but it requires a bit more processing before it is useable. We filter newlines, URLs, mentions and non-alphabetic from the contents of the Tweets. Remove these attributes from texts is done to reduce noise, standardize the text, and also just to make the data more readable. We would not want the clustering algorithm to cluster Tweets together based on how many of these properties they contain. Furthermore, these elements of text do not hold any sentimental value, and having them there could only interfere with the sentiment analysis. This formatting of the data thus helps ensure consistency and makes the process of clustering and analysing sentiment simpler.

3.3.2 Case Study Specific Parameters

Throughout our general approach, there are several instances when a parameter had to be changed depending on the data it is applied to. The specific instances of this were the list of stop words we had to define for the Tokenization of Topics using CountVectorizer, the language parameter we had to set during the Creation of BERTopic Model, and finally the specific sentiment model we want to use for our Sentiment Analysis.

The List of Stop Words

Because all of our data consists of Tweets mentioning "ChatGPT", there is a high chance that the clustering algorithm will create one big cluster for the topic "ChatGPT". This is not useful to us, as we would like to view the sentiment on smaller issues within this large topic. To prevent this from happening, we can pass a list of words we want to leave out of the topics to the `stop_words` parameter. The list of stop words we create can be subdivided into the following three sublists:

- One list with words related to AI and ChatGPT, so we do not get one big cluster.
- One list with sentiment related words, as running the sentiment analysis over a cluster based on positive words like "good" and "amazing" is not going to give any insightful results.
- One list of general stop words with words like "the", "and", "I" that appear very often in English, but do not give a lot of useful information on the topic.

The list of AI words and sentiment words have both been created manually by running the algorithm several times, and checking if there were any topics created with words that fall in either category. The list of general

stop words are taken from a list created by Sean Bleier [1]. While not being completely extensive, these three lists together cover a large portion of the words we would like to ignore in the formation of the topic.

The Language Parameter

Since the dataset was gathered using the command `(chatgpt OR ChatGPT) lang:en -is:retweet -is:reply`[1], all Tweets should be in English. This means that setting the `language` parameter of the BERTopic model to "English", will give us a better result. This is because having the language model be pre-trained on the same language as the test data will lead to better clustering.

The Sentiment Model

To perform the sentiment analysis, we utilize the pre-trained `cardiffnlp /twitter-roberta-base-sentiment-latest`[2] model from the Hugging Face library. This model is part of the TimeLMs models, with this particular model being specialized in sentiment analysis.

Since our data set contains Tweets of the first month since the launch of ChatGPT, or more specifically from 30/11/2022 to 31/12/2022[1], it would be best to have a sentiment analysis model that is also trained on Tweets within this time period. However, at the time this research was performed, there still was not a trained model available for that specific time period. That is why we used the latest available model instead, which at the time of this research was trained on tweets from January 2018 to December 2021, as it is the closest to our desired time frame.

Chapter 4

Results

4.1 Clustering and Sentiment Findings

The clustering part of the algorithm resulted in the creation of 258 topic clusters. This includes the -1 cluster, containing all Tweets that BERTopic was not able to find a matching topic for. After the removal of this cluster, we will thus be left with 257 topics to analyse the sentiment for. A graphical representation of the topic clusters can be seen in Figure 4.1.

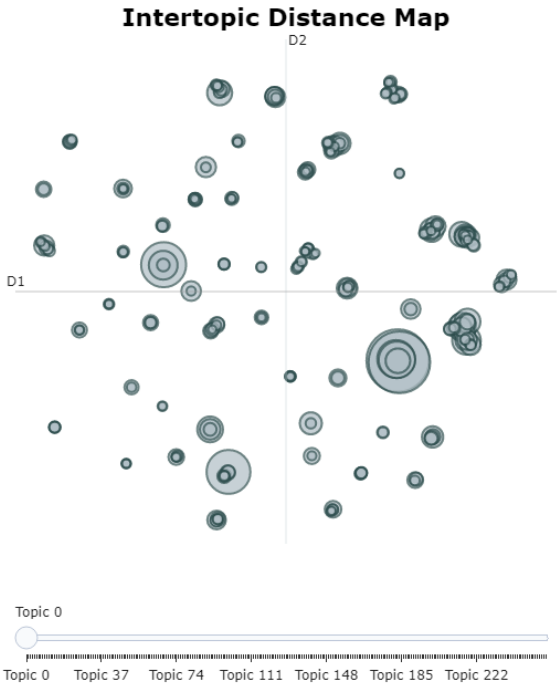

Figure 4.1: Visualization of Topic Clusters Generated by the Clustering Algorithm

The topics of the Tweets can be divided into 2 categories, Tweets about possible applications for ChatGPT, and Tweets providing commentary on societal impact ChatGPT has. We will discuss these two categories separately.

4.1.1 Sentiments on Specific Applications for ChatGPT

In this subsection, we explore the sentiments expressed in Tweets about various applications for ChatGPT. Many possible applications were discussed, but we chose to present only some of the most notable ones for the sake of brevity. These notable topics are related to cooking, essay writing, healthcare, image generation, maths, poem writing, programming assistants, and search engines. Each topic is represented by a separate pie chart, providing insights into the sentiments associated with these application domains.

Cooking

In Figure 4.2 the sentiments expressed in Tweets related to cooking and coming up with recipes are represented. People are still mostly positive about how easily recipes can be generated to suit your specific needs, but there is also some negativity, with people expressing that not all the recipes are all that great.

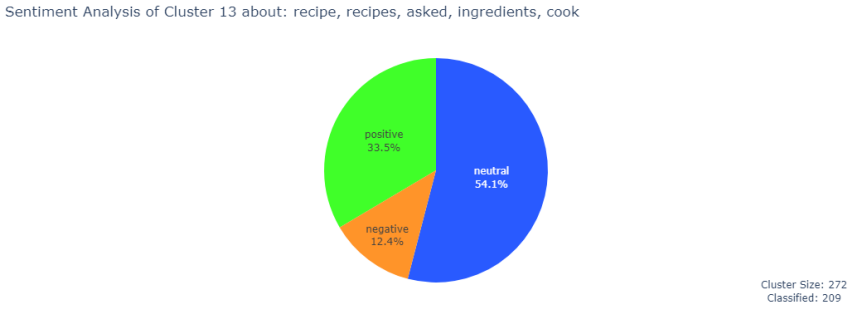

Figure 4.2: Sentiment Analysis Results for Tweets on ChatGPT's Application in Cooking

Essay Writing

The sentiments regarding students using ChatGPT for essay writing are visualized in Figure 4.3. This topic is a bit more controversial, as there is not an all that big difference in the percentage of positive and negative sentiments. While some people admire ChatGPT's ability to generate convincing

essays, many people are also concerned about the ways this will be used to cheat the current school system.

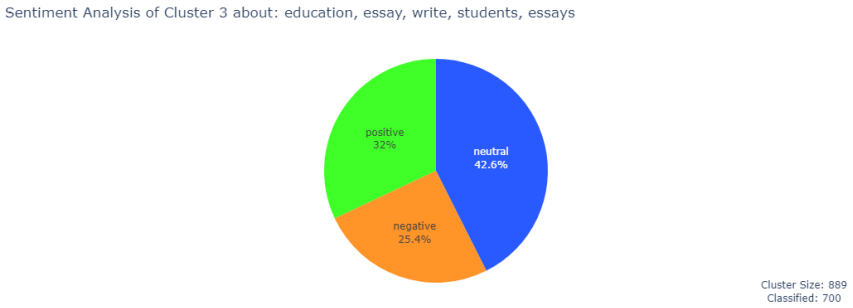

Figure 4.3: Sentiment Analysis Results for Tweets on ChatGPT's Application in Writing Essays

Healthcare

Then, Figure 4.4 displays the sentiments expressed in Tweets regarding ChatGPT's potential applications in the medical field. Here it is again clearly visible that people are far more positive about the use of ChatGPT in this field, than they are negative.

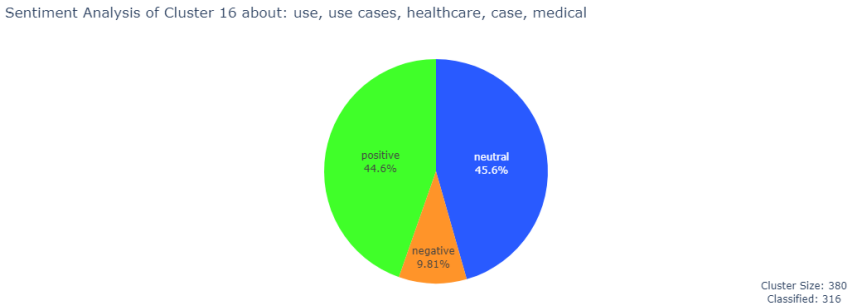

Figure 4.4: Sentiment Analysis Results for Tweets on ChatGPT's Application in Healthcare

Image Generation

The sentiments expressed in Tweets discussing ChatGPT's image generation capabilities are depicted in Figure 4.5. This is again a pretty controversial

topic, with a slight edge towards the positive side. Similar to writing essays, people are impressed by what can be done, while others are worried about the dilution of real, man-made art.

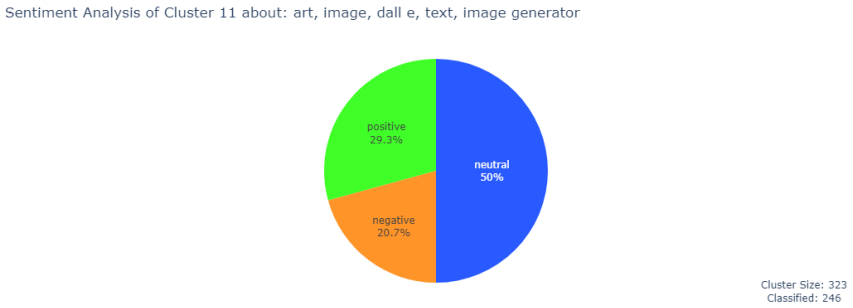

Figure 4.5: Sentiment Analysis Results for Tweets on ChatGPT's Application in Generating Images

Maths

The sentiments expressed in Tweets related to ChatGPT's applications in mathematics are visualized in Figure 4.6. Maths being one of the topics where the vast majority of Tweets tend to be negative about the use of ChatGPT. This is mainly due to the amount of answers ChatGPT gets wrong. In an exact science like maths, even small errors can lead to entirely inaccurate solutions.

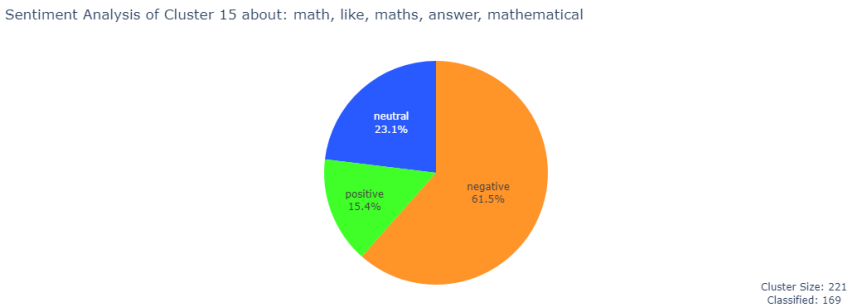

Figure 4.6: Sentiment Analysis Results for Tweets on ChatGPT's Application in Maths

Poem Writing

Figure 4.7 displays the sentiments expressed in Tweets discussing ChatGPT's application in poem writing. The majority of Tweets here are just showcasing poems that ChatGPT wrote, these Tweets are mostly neutral in sentiment, as the poems are often included as images. On the more opinionated ends of the spectrum, most people seem to like the poems that ChatGPT writes, while a few complain about satisfied with the amount of control they have on having the poem be in a certain style.

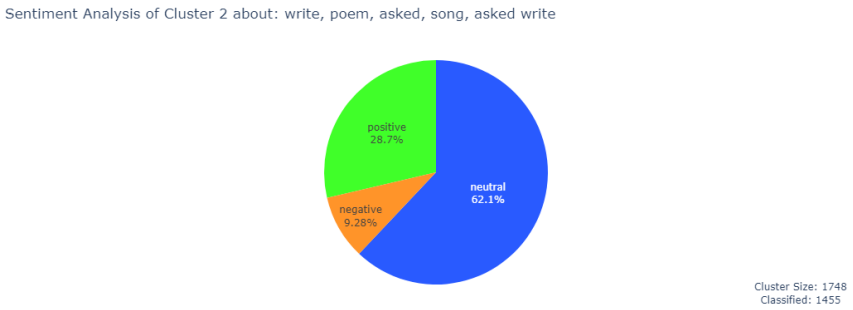

Figure 4.7: Sentiment Analysis Results for Tweets on ChatGPT's Application in Writing Poetry

Programming Assistant

In Figure 4.8 represents the sentiments expressed in Tweets related to ChatGPT's role as a programming assistant. Again, the neutral Tweets contain a lot of images of the code it wrote. Disregarding these, the vast majority of people like are impressed with the code generated, and also praise how much ChatGPT is able to speed up their workflow. On the other hand, the code being generated definitely is not perfect, as quite a large portion of Tweets are complaining about bugs in their code.

Search Engine

Lastly, Figure 4.9 shows the discussing on using ChatGPT as an alternative to a search engine like Google. This is another topic in which both sides of the argument are prevalent. While a majority seems to believe ChatGPT can be a good alternative to search engines like Google, others comment on issues like ChatGPT being slow when server load is high, or its subpar performance in languages other than English.

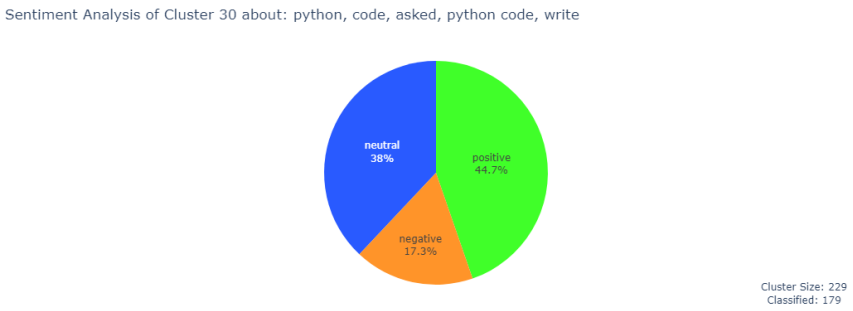

Figure 4.8: Sentiment Analysis Results for Tweets on ChatGPT's Application as a Programming Assistant

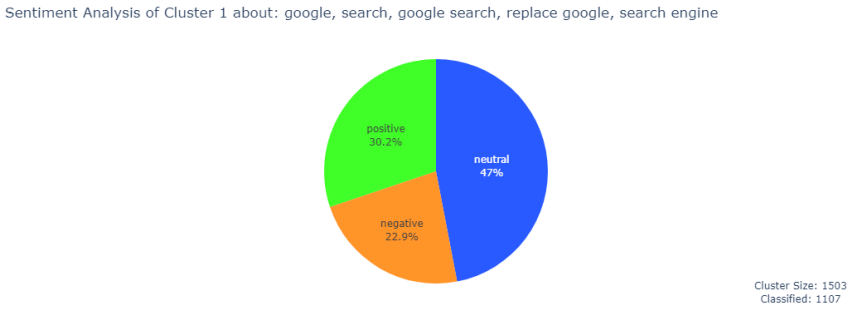

Figure 4.9: Sentiment Analysis Results for Tweets on ChatGPT as a Search Engine

4.1.2 Sentiments on ChatGPT in the Broader Scope

In this subsection, we delve into the sentiments expressed in Tweets regarding the societal impact of ChatGPT. Understanding public perceptions and attitudes towards ChatGPT's influence on society is crucial for assessing its potential implications. We focus on the following key subtopics: environmental impact, ethics, and the impact on the job market. Each subtopic is again accompanied by a separate analysis, shedding light on the sentiments associated with these aspects.

Environmental Impact

Figure 4.10 illustrates the sentiments expressed in Tweets discussing ChatGPT's impact on climate-related issues. A very controversial issue, with the slight majority of people having a negative view on the environmental impact of using ChatGPT, while the other side expresses hopefulness that using ChatGPT will allow scientist to come up with better methods of reducing climate pollution.

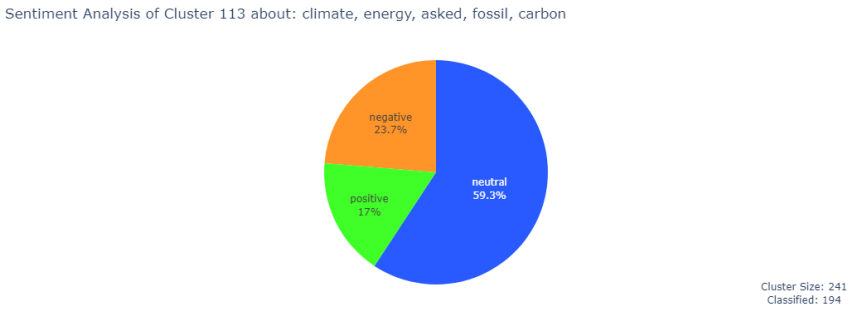

Figure 4.10: Sentiment Analysis Results for Tweets on ChatGPT's Impact on the Environment

Ethical Concerns

The sentiments expressed in Tweets regarding the ethical concerns surrounding ChatGPT are visualized in Figure 4.11. While the majority of Tweets on this topic are neutral, a big portion shows to have worries about the ethical implications of ChatGPT, being worried that people will blindly follow psychological or medical advice without being certain it will work or if ChatGPT has proper morals.

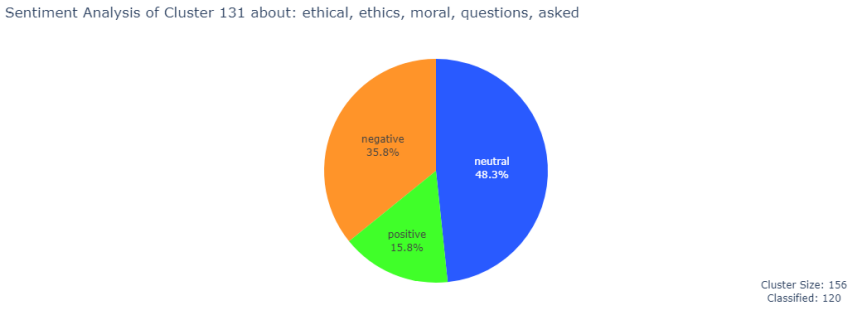

Figure 4.11: Sentiment Analysis Results for Tweets on Ethical Concerns Regarding ChatGPT

Impact on the Job Market

Finally, figure 4.12 showcases the sentiments expressed in Tweets discussing the potential impact of ChatGPT on jobs and employment. While a larger portion seems to be worried that this new technology will cause people to lose their jobs, a significant portion is optimistic about the new job opportunities it will create.

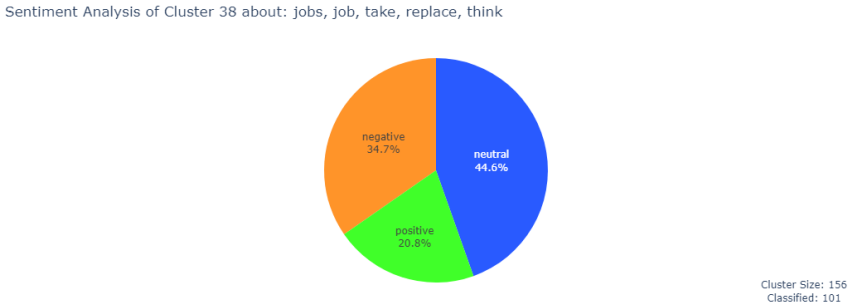

Figure 4.12: Sentiment Analysis Results for Tweets on ChatGPT's Impact on Jobs

4.2 Invalid Results

Unfortunately, not all the results we get are as insightful as the examples given above. The method does not always result in insightful topics and sentiment. These instances can be further subdivided into bigger problems,

or smaller problems that are only issues in some cases.

4.2.1 Problematic Results

The following issues, are problematic to the overall accuracy and usefulness of the program. These problems are related to unwanted topics, Tweets labelled to the wrong sentiment, and duplicate topics. Fixing these problems is important to the general performance of the algorithm, and will be one of the main areas for further research.

Unwanted Topics

Some of the topic sentiments are extremely one-sided, not because people agree so much on a topic, but because the topic is not valid. As mentioned in the Tokenization of Topics using CountVectorizer section, our list of stop words is not extensive. Because of this, we will sometimes still get clusters based on sentiment related words, ChatGPT related words, or just general stop words.

An example of this is Figure 4.13, which shows the sentiment of Tweets about ChatGPT being a game changer. The Tweets in this cluster are mostly going to be positive, because they were clustered on words that are generally considered to be positive. Clusters like these are not going to give us any insightful information.

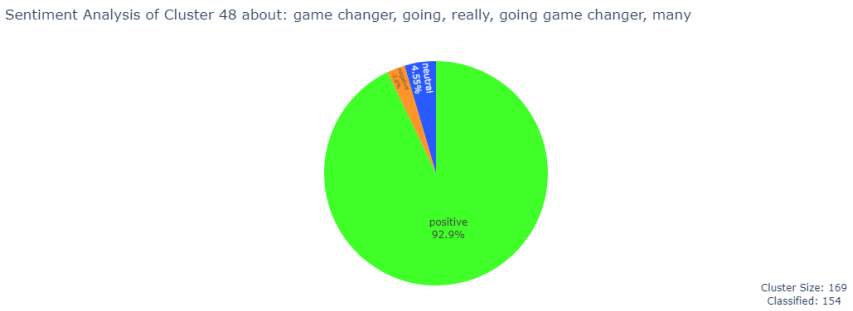

Figure 4.13: Example of an Unwanted, Sentiment Related Cluster

Wrong Sentiment Attribution

Sometimes, the sentiment analysis will label a Tweet to the wrong sentiment. While we did have a filter for removing Tweets of which the model was uncertain if it got classified correctly, sometimes the algorithm gains false confidence due to a confusing use of words. Naturally, having Tweets that

belong to the wrong class, will have a bad influence on the accuracy of our results.

For example, as can be seen in Figure 4.14 which is about the cost of ChatGPT, one of the Tweets classified as negative says: "make chatgpt paid I will pay for this shit". This Tweet states that ChatGPT is so good, that if it would cost money, this person would gladly pay it. However, due to the word "shit" being used, which is generally negative, it got labelled incorrectly.

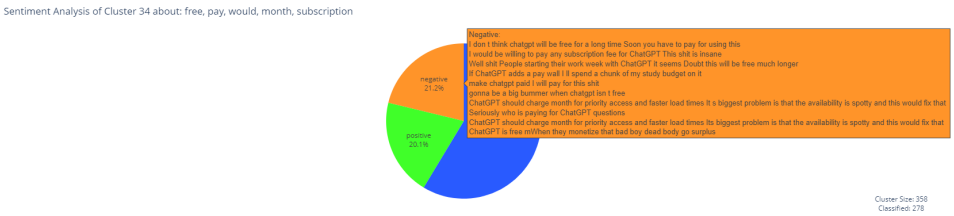

Figure 4.14: Example of a Wrong Sentiment Attribution

Duplicate Topics

Finally, an issue that is a bit less damaging to the overall results, but still not ideal, is that there are several instances of two or more topic clusters being about the same subject. For example, as can be seen in Figure 4.15, we have a topic cluster about creating images. This is very similar to the topic we had in the Image Generation example, which was also about image generation. Ideally, these clusters about the same subject would be merged into one bigger cluster, to give a more comprehensive overview of the public opinion on the matter.

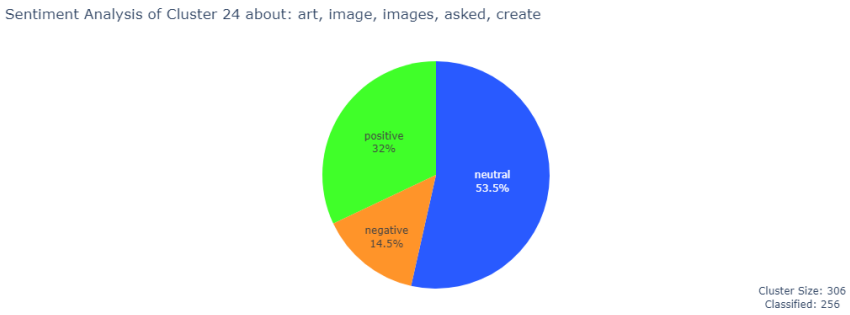

Figure 4.15: Example of a Duplicate Topic

4.2.2 Potential Problems

The issues presented in the following section will be less damaging to the overall performance of the algorithm. These problems are related to the inclusion of duplicate tweets, incoherent topics, and similar topics. Depending on the use case, these things may not even be problems at all. However, to allow for all use cases of an unsupervised topic-based sentiment analysis tool, at least an option should be made available to avoid them.

Duplicate Documents

In certain instances, we encountered a significant number of duplicate tweets within the data. In Figure 4.16 for instance, we see the Tweet "This AI chatbot is dominating social media with its frighteningly good essays" multiple times. This largely happens because of Twitter's Retweet feature, but may also be caused by bot accounts spamming the same Tweet, or multiple people reporting on the same news headline, for example. Whether it is necessarily an issue that multiple instances of the same Tweet is up for debate.

A retweet expresses people agreeing with a statement, and if more people think share this opinion, it should be included more often. However, some people might want to eliminate bots, or only look at unique opinions. For these use cases, a method of removing duplicate data should be implemented.

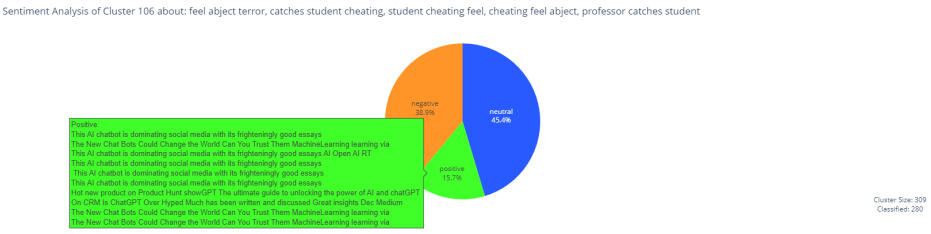

Figure 4.16: Example of Duplicate Tweets within a Topic

Incoherent Topics

As was already expressed in the Weighting Tokens & Topic Representation and Retrieving Key-Phrases sections, we did not have enough time to properly experiment with the naming of topics in this research. Because of this, some of the words that are supposed to represent a topic, do not really do a good job of telling what the topic is about.

For example, the graph shown in Figure 4.17 has the title: "Sentiment Analysis of Cluster 27 about: thing, also, actually, kinda, time". These words do not at all indicate that the Tweets are about finding ChatGPT scary (also another instance of Unwanted Topics).

This is not a huge problem, as we are able to look at the individual Tweets within the cluster to identify the topics ourselves, however, this goes against the goal of being an unsupervised algorithm.

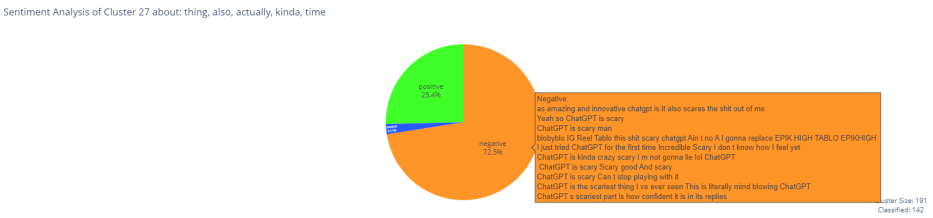

Figure 4.17: Example of an Incoherent Topic Representation

Similar Topics

The last issue is similar to the issue described in the Duplicate Topics section, but in a more specific sense. In Figure 4.18, we see a cluster about writing Haikus, a specific form of Japanese poetry. We already had a cluster about writing poetry in the Poem Writing section.

Whether writing Haikus should be included within this bigger topic of writing poetry, or left as a standalone topic, can be up for debate. Regardless, having the extra option to have control over when to merge subtopics, and when not, would be a feature worth implementing.

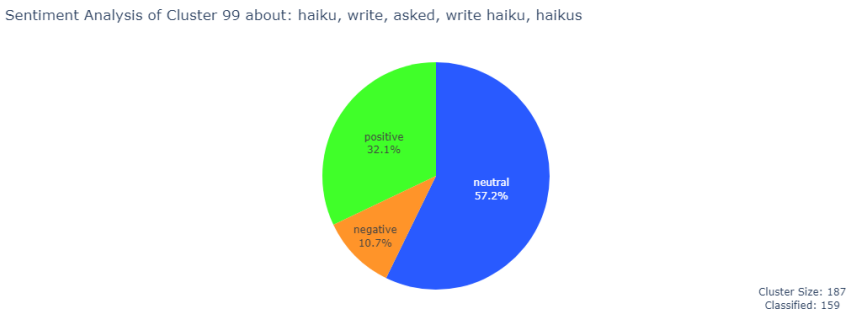

Figure 4.18: Example of a Topic, Similar to Another

Chapter 5

Discussion

5.1 Discussion

The clustering and sentiment analysis results provided valuable insights into the public perception and sentiment regarding various applications of ChatGPT, and its broader societal impact, but also displayed the areas in which the method proposed can be further improved. In this section, we will delve into our findings, highlighting the key observations as well as the potential areas for improvement in further research.

5.2 The Case Study

We will first be discussing the findings of the case study that motivated this thesis project.

5.2.1 Discussion: Sentiments on Applications for ChatGPT

The sentiment analysis of Tweets on various applications for ChatGPT revealed that the majority of fields are optimistic. Three of the topics had a clear positive sentiment, these topics being about cooking, healthcare, and programming assistant. Of the remaining topics, essay writing, image generation, and search engines were a bit controversial, with a slight edge towards being positive. Only in the field of mathematics, the sentiment was mostly negative.

The majority of the complaints about applications of ChatGPT seem to be coming from the areas where mistakes are most noticeable. In fields such as mathematics and programming, even a small mistake can lead to a wrong answer or not working code. ChatGPT, at this point, just is not accurate enough to be used for these more exact sciences. Also, when used as a search engine, many complaints are about ChatGPT being slower, and less accurate in languages other than English, than a traditional search engine would be.

These are again limitations of the software. In more abstract topic areas, like writing poetry or cooking, these small mistakes are less noticeable, and people tend to think a bit more positively of ChatGPT's capabilities.

However, while many are criticising ChatGPT for the errors it makes, there are others expressing optimism about the future of ChatGPT. These people acknowledge the faults ChatGPT makes, but also keep in mind that this technology is fairly new, and still being worked on. They are hopeful that in a few years, there will not be any false outputs any more, and they are thinking about all the things that would be possible once this software is more developed.

It is interesting to note some of the contradictions here. People are negative towards ChatGPT being used in the field of maths, due to it giving incorrect answers. But, while it would be reasonable to assume that for something as delicate as healthcare, providing correct answers would be even more important, the opinions on this issue are much more positive.

5.2.2 Discussion: Sentiments on ChatGPT's Social Impact

The sentiment analysis of Tweets on ChatGPT in a broader scope lead to more controversial issues. The opinions on all topics here are also much more pessimistic, with the sentiments on ethical concerns and the impact on the job market being mostly negative. The discussion on the environmental impact this technology will have was a bit more divided, but still also leans towards a mostly negative overall opinion. This is a stark contrast to the mostly optimistic opinions of ChatGPT's possible use cases.

It seems these broader discussions on societal issues with generative AI are much more controversial than the discussions on applications, because there are much more perspectives to consider on an issue like ethics, than on the ability of ChatGPT correctly being able to answer mathematical problems. Different people have different morals to live by, and because of this they will all look at the ethical implications of these chatbots differently, while with a programming assistant the code simply works, or not.

It also appears that most of the negative sentiment towards ChatGPT comes from a fear of what might happen, rather than a problem with the software itself. People are scared of losing their jobs due to their work being automated, scared that someone will harm themselves or others because of bad advice, or scared of the increase in energy consumption constant use of this software will have. These are not problems of technology itself, but rather side effects of the existence of this software.

On the other side of the spectrum, we have the people who are hopeful about the prospects ChatGPT will bring with it. These chatbots are energy intensive, but they may aid researchers in finding ways of reducing emissions. Some jobs may disappear, but new ones will take their place. These hopeful arguments, however, are being outnumbered by the more fearful ones.

5.3 Discussion: Limitations and Future Work

The results also displayed the shortcomings of the approach we developed, some problematic, some only inconvenient. The majority of these problems stem from the topic identification and clustering part of the program. Of the six issues found, four of them arise in this step, two of which are considered to be a big problem. The issues that need fixing the most are related to the existence of unwanted topics due to our list of stop words not being extensive, and the existence of multiple topics about the same subject. The other problems are related to the topic representation and not having control over the level of detail of the topics.

The problem of the unwanted topics, as well as the topic representation, can likely be solved quite easily. As mentioned in the Weighting Tokens & Topic Representation section, we did not have enough time for this research to properly implement the last two steps of the BERTopic algorithm. If we were to implement the last two steps in the future, these issues are likely resolved, or at least mitigate it somewhat. With the weighting of the tokens it should be possible to give stop words, not in the stop word list, a lower weight. This will remove the unwanted topics due to our list of stop words not being extensive. The final topic representation step should help summarize better what a cluster is about, and thus solve the problem of incoherent topics.

Solving the problem of duplicate and similar topics may be a bit more difficult. If we are lucky, this problem is only caused by some parameters of one of the BERTopic steps being off. We chose to use very densely packed clusters, as we did not want any vague topics that were not about any specific topic. However, it might be that the clusters we created are so dense, that things that one topic gets split further into more subtopics. If this is the case, tweaking some of the parameters that are involved in how dense the clusters are, like UMAP's `min_dist` and `n_components`, might be enough to mitigate this problem.

If it turns out that just tweaking the parameters is not enough, then the BERTopic model should be modified in such a way that the user has control over when to merge certain topics or not. An attempt at this has been made before by Ariel Ibaba [11]. Unfortunately, again due to time constraints, this method has not been tested during this research, and we are not able to say whether this solution works or not. Nevertheless, the version it was built on is an outdated version of BERTopic from 2021, and it has not been updated since then. Because of this, it may be that it does not work that well, and a more up-to-date method has to be developed. An interactive method using visual analytics would be an interesting direction for future research.

Just because the majority of the problems arise in the topic creation, does not mean that the sentiment analysis is without faults. Some of the documents were classified as the wrong sentiment, which interferes with the

final sentiment distribution. This may in part be because we had to use a model that was trained on Tweets from a different time period than the Tweets in the test data, like described in Sentiment Analysis, but it can also be due to these models not being entirely accurate in general. Fixing this issue will be a matter of training better sentiment analysis models.

Then the problem of duplicate documents should be pretty easy to solve. Just adding a simple function that removes duplicate entries can already do a lot of the work necessary here. Sometimes two documents are not identical, but are eerily similar. To remove these cases, a more advanced method of removing very similar documents has to be implemented, however such methods already exist as well. Just integrating a function like this will give the user the choice to remove duplicates when wanted.

A final suggestion, for future research, would be to implement a method of measuring how the sentiment changes over time. Implementing this would be especially interesting for seeing how the more controversial discussions evolve over time, however, this regrettably was outside the scope of this research. Creating such a feature should be doable, as the exact time the post was created is often saved in the metadata. By also using this data, we should be able to split the topic clusters into smaller groups based on their time frame, apply sentiment analysis over all these timeframes, and then graph the prevalence of each sentiment over time. Ideally, the size of the timeframe could also be set as a parameter, to be either years, months, or weeks for example.

Chapter 6

Conclusions

In conclusion, this paper addresses the impact of social media on public discourse and presents a solution to the challenges of analysing vast amounts of non-uniform data. Our proposed algorithm leverages the capabilities of LLMs to cluster social media posts based on topics and perform sentiment analysis.

Throughout this study, we explored key concepts and methodologies. We examined Transformer models, which process large text datasets through self-attention mechanisms. We introduced cluster analysis, similarity measures, and the HDBSCAN algorithm for hierarchical density-based clustering. To handle high-dimensional data challenges, we explored sentence embeddings and dimensionality reduction techniques. Additionally, we discussed the steps in sentiment analysis and the role of machine learning models.

The methodology for analysing the sentiment of documents in separate topics involved data preparation steps, such as document filtering and topic grouping using the BERTopic algorithm. Sentiment analysis was performed using a pre-trained model, with post-processing techniques mapping documents to sentiment labels. Finally, the sentiment analysis results were visualized through informative pie charts.

The final results of our case study of applying this algorithm on Tweets about ChatGPT were able to shed light on public perception and sentiment regarding ChatGPT's applications and broader societal impact. We observed that people tend to think positively of the application of ChatGPT in more abstract fields, while in the more exact sciences people are more negative as they think the software is not accurate enough. On broader discussions about the societal impact of this new technology, some people are hopeful about the future possibilities this software will bring with it, but the majority are still fearful for the negative implications on the world.

While the algorithm created was able to give us insights into the public opinions on many topics, as it was intended, there are still some limitations.

These limitations are mainly connected to the identification and clustering of topics part of the paper, partly because of a limited implementation of the BERTopic model due to time constraints, but also because of some missing features in this model. The sentiment analysis also was not completely accurate yet, and better models should be trained in the future to improve on this. Aside from these issues, some lacking features were also identified, such as an option for removing duplicate documents, and analysis of sentiment of time.

Bibliography

[1] Sean Bleier. Nltk's list of english stopwords, 2010. https://gist.github.com/sebleier/554280 (visited on 15/05/2023).

[2] Jose Camacho-Collados. cardiffnlp/twitter-roberta-base-sentiment-latest, 2023. https://huggingface.co/cardiffnlp/twitter-roberta-base-sentiment-latest (visited on 21/03/2023).

[3] Ricardo J. G. B. Campello, Davoud Moulavi, and Joerg Sander. Density-based clustering based on hierarchical density estimates. In Jian Pei, Vincent S. Tseng, Longbing Cao, Hiroshi Motoda, and Guandong Xu, editors, *Advances in Knowledge Discovery and Data Mining*, pages 160–172, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[4] KENNETH WARD CHURCH. Word2vec. *Natural Language Engineering*, 23(1):155–162, 2017.

[5] Jorge Carrillo de Albornoz, Julio Gonzalo, and Enrique Amigó. Replab: An evaluation campaign for online monitoring systems. In *Information Retrieval Evaluation in a Changing World*, pages 487–510. Springer, 2019. https://link.springer.com/chapter/10.1007/978-3-030-22948-1_20.

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

[7] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Knowledge Discovery and Data Mining*, pages 226–231, 1996.

[8] Maarten Grootendorst. Bertopic - advanced customization, 2022. https://colab.research.google.com/drive/1ClTYut039t-LDtlcd-oQAdXWgcsSGTw9?usp=sharing#scrollTo=AXHLDxJdRzBi (visited on 20/04/2023).

[9] Maarten Grootendorst. Bertopic: Neural topic modeling with a class-based tf-idf procedure. *arXiv preprint arXiv:2203.05794*, 2022.

[10] Maarten Grootendorst. Using whisper and bertopic to model kurzgesagt's videos, 2022. `https://towardsdatascience.com/using-whisper-and-bertopic-to-model-kurzgesagts-videos-7d8a63139bdf` (visited on 20/04/2023).

[11] Ariel Ibaba. Atoti bertopic, 2021. `https://github.com/atoti/BERTopic` (visited on 20/04/2023).

[12] Daniel Loureiro, Francesco Barbieri, Leonardo Neves, Luis Espinosa Anke, and Jose Camacho-collados. TimeLMs: Diachronic language models from Twitter. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 251–260, Dublin, Ireland, May 2022. Association for Computational Linguistics.

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

[14] Omar Espejel Nils Reimers. sentence-transformers/all-minilm-l6-v2, 2022. `https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2l` (visited on 19/05/2023).

[15] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up? sentiment classification using machine learning techniques, 2002.

[16] 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.

[17] Nils Reimers. Sentence transformers - pretrained models, 2022. `https://www.sbert.net/docs/pretrained_models.html` (visited on 19/05/2023).

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

[19] Matthew Schwartz. Quickly find common phrases in a large list of strings, 2020. `https://dev.to/mattschwartz/quickly-find-common-phrases-in-a-large-list-of-strings-9in` (visited on 15/05/2023).

[20] Tan, Steinbach, and Kumar. *Introduction to Data Mining*. Addison-Wesley, 1 edition, 2005.

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30*. NeurIPS, 6 2017.

[22] Suzan Verberne. Het succes van bert en huggingface - taaltechnologie in 5 regels code. *DIXIT*, pages 3–4, 2022.