Analyzing Online Sentiments and Topics in the Israel-Palestine Conflict: A Reddit Discourse Study

Greg Forkutza
Student ID: 400277514

14 December, 2023

1 Abstract

This report explores online discourse trends and dynamics on Reddit, specifically focusing on the Israel-Palestine conflict. Sentiments and topics within top Reddit comments from the r/news subreddit over the course of a year are extracted from the Reddit API in order to provide insights into public opinion dynamics and contribute to the understanding of digital narrative formation in social media platforms. The methodology incorporates Natural Language Processing techniques, including text embeddings using BERT, dimension reduction via PCA combined with t-SNE and UMAP, and clustering methods like K-means and Gaussian Mixture Models. Additionally, Latent Dirichlet Allocation is employed for topic analysis, and sentiment trends are assessed using VADER. The results indicate varying sentiment trends and discussion focus. However fine tuning of the algorithms used and deeper qualitative analysis of the results are required in order to make more meaningful analysis of the data.

2 Introduction

The advent of social media platforms has revolutionized the way information is disseminated and discussions are held, particularly on controversial global issues. Among these platforms, Reddit stands out as a online center of diverse opinions and dicussions, making it a great source for understanding public sentiment and discourse. We look into online discourse on Reddit, with a specific focus on the Israel-Palestine conflict.

The Israel-Palestine conflict, marked by its long history and complex geopolitical implications, has been a subject of intense debate and varied opinions. We aim to analyze the nature of discourse surrounding this topic on Reddit, particularly in the r/news subreddit. By employing advanced Natural Language Processing techniques, we seek to uncover underlying sentiment trends, thematic focuses, and temporal shifts in the discussion.

Through the use of BERT for text embedding, we try to capture linguistic features of Reddit comments. Dimension reduction techniques, including PCA combined with t-SNE and UMAP, are applied to manage the high-dimensional nature of text data. To further distill meaningful patterns from the data, clustering methods such as K-means and Gaussian Mixture Models are utilized. The analysis is complemented by LDA for topic extraction, offering insights into the prevailing themes in the discourse. Sentiment analysis provides an additional layer of understanding by evaluating the emotional tone of the comments. By analyzing sentiment trends over time, we aims to shed light on how public opinion and discussion focus have evolved in relation to the Israel-Palestine conflict.

3 Data Engineering

3.1 Workflow and Modular Development Practices

A primary objective of this project was to establish an end-to-end data pipeline featuring modular and robust function support. By designing each function in the data processing and analysis pipeline as an independent module, the system's scalability is ensured, facilitating potential integration with advanced tasks such as fine-tuning pre-trained models. Although time constraints limited the

development of data streaming capabilities necessary for fine-tuning models like Base-BERT for more precise political bias and sentiment detection, the foundational codebase is prepared for such advancements.

The project's architecture is characterized by a series of functions, each performing specific tasks within the processing and analysis phases. Comprehensive wrapper functions in each section amalgamate these individual functions into unified operations, exemplifying the system's modularity. In Section 5.3, the modular approach is illustrated through the optimization of two dimension reduction and clustering methods, along with their associated parameters, across numerous datasets. Section 7 discusses the sequential tasks required to scale the project for future developments and enhancements.

3.2 Reddit API Wrapper

In this project, a minimalist Reddit API wrapper was developed for R, facilitating interaction with Reddit's comment data. The process began with establishing a dedicated Reddit user account, serving as the primary interface with the Reddit ecosystem. Subsequently, an application was registered with Reddit, a standard requirement for API access. The application utilizes OAuth 2.0 protocol for authentication with the Reddit API, ensuring secure authorization without directly exposing user credentials. This implementation was achieved using the `httr` package, a prominent tool for managing HTTP GET requests and authentication protocols [Wickham, 2022].

The API wrapper comprises several key functions, delineated with the prefix `get_`. The `get_url` function constructs a JSON search query based on user-defined parameters such as keywords, timeframe, subreddit selection, and a ranking parameter for retrieved comment threads, yielding a vector of permalinks to the relevant threads. The `get_content` function utilizes these permalinks to extract comments in JSON format through GET requests, returning a nested list structure in R. The `get_comments` function simplifies this complex nested list into a structured data frame, including variables such as comment text, score, and reply/children dependencies. Lastly, `get_data` serves as a comprehensive wrapper function, allowing users to specify search queries and receive an organized data frame. This function enhances the scalability of the project, enabling integration into automated, continuous, or batch data processing streams, potentially employing technologies like Kafka. Their definitions can be seen in section 9.1.

4 The Data

A structured query was initiated using the search endpoint of the Reddit API [Red, 2023]. The query parameters were configured to retrieve the top 50 posts from the r/news subreddit, focusing on posts with the highest upvote-to-downvote ratio over the past year. The keywords employed in the search included "Gaza," "Israel," "Palestine," "Jerusalem," "West Bank," and "Hamas." This comprehensive search yielded a dataset encompassing a total of 22,723 comments, each characterized by 11 distinct variables.

The data for each comment includes a unique identifier, the Reddit username of the author, and the comment's score, represented as the upvote-to-downvote ratio. Additionally, the dataset details the number of direct replies (children) and the total number of replies at all levels (descendants) for each comment. The 'children' count refers to immediate responses to the comment, whereas 'descendants'

encompass all nested replies, thereby providing a depth-wise account of the conversation. Other critical data elements encompass the date and time of the comment's posting, the title of the original post to which the comment was made, the name of the subreddit, and the specific path to the Reddit endpoint associated with the comment.

The dataset exhibits a wide range of engagement levels, with the smallest post containing 71 comments and the largest amassing 1,238 comments. For public accessibility and further examination, the collected data is hosted on the author's GitHub repository [Forkutza, 2023].

4.1 Pre-Processing

The dataset comprised a list of 50 elements, each representing a distinct comment thread from Reddit. The initial step in preprocessing involved extracting the original posting date of each thread. Subsequently, these threads were arranged in chronological order.

Next we focused on refining the textual content of each comment,in order to optimize the outcomes of subsequent Natural Language Processing (NLP) techniques. This involved the removal of special characters and numerical digits, and the conversion of all uppercase letters to lowercase. Additionally, comments that were either deleted or removed from the platform were excluded from the dataset.

Tokenization and lemmatization of the comments were not performed prior to applying the BERT model for text embedding. This is due to BERT's inherent capability to automatically execute these tasks. However, post-embedding, for the purpose of topic analysis via Latent Dirichlet Allocation (LDA) as detailed in Section 5.4, the comments were lemmatized and tokenized. This was accomplished using the `tm` package [Feinerer and Hornik, 2023].

5 Methods

In this study, the methodology was structured to analyze online discourse through Reddit comments. The initial step involved consolidating 50 comment threads into ten groups, each containing five threads arranged in chronological order.

We utilized the Bidirectional Encoder Representations from Transformers (BERT), a language model grounded in the transformer architecture. BERT's embeds text documents into a high-dimensional vector space. The embedding process was applied to each group of five comment threads, resulting in ten distinct lists of text embeddings.

To explore the temporal dynamics of the dataset, these grouped embeddings were further compiled into ten matrices. Each matrix represented an aggregation of embeddings corresponding to five chronologically ordered comment threads. Dimensionality reduction was achieved through two techniques: Principal Component Analysis combined with t-distributed Stochastic Neighbor Embedding and Universal Manifold Approximation and Projection. The reduced embeddings then underwent clustering using two distinct methods: K-means and Gaussian Mixture Models. This approach yielded four combinations of dimension reduction and clustering: PCA with t-SNE and K-means, PCA with t-SNE and GMM, UMAP with K-means, and UMAP with GMM.

Each combination was fine-tuned for optimal performance; perplexity parameters were adjusted for PCA with t-SNE, and the number of clusters was optimized for all four methods. The effectiveness

of each method was evaluated using silhouette scores, facilitating the selection of the most suitable clustering configuration for each subset of comment threads. The resultant data included the cluster labels for comments within each chronologically ordered subset.

Further analysis involved sentiment scoring for individual comments in each subset, employing sentiment analysis techniques to gauge the emotional tone of the discourse. Additionally, Latent Dirichlet Allocation (LDA) was utilized to identify prevalent topics within each subset, assigning topic labels to comments and extracting the most significant words associated with these topics. Consequently, the final output consisted of ten dataframes, each representing five comment threads. These dataframes encapsulated the clustering configurations, sentiment scores, and topic assignments for the comments, along with a corresponding dataframe detailing the top words in each identified topic and the dtm object (which was not used further in this report).

In the following subsections, the statistical, computational, and practical nuances of these methods are elaborated on in order to provide better insight into the analytical framework employed in this research.

5.1 Bidirectional Encoder Representations from Transformers (BERT)

Transformers represent a paradigm shift in how algorithms process sequential data. Unlike their predecessors, Recurrent Neural Networks and Long Short-Term Memory networks, transformers forgo sequential processing in favor of parallel computation, which significantly enhances computational efficiency and model performance in large datasets. The core innovation of transformers lies in the 'attention mechanism', which allows the model to weigh the significance of different parts of the input data irrespective of their position. This mechanism enables the model to capture complex dependencies and relationships within the data.

Bidirectional Encoder Representations from Transformers, or BERT is based on the transformer architecture and is designed to understand the context of a word in a sentence from both directions, which is an advancement over previous models that processed text unidirectionally. This bidirectionality allows BERT to capture a more nuanced and comprehensive understanding of language context.

BERT's basic functionality is as a pre-trained model. It is initially trained on a large corpus of text, using tasks like Masked Language Modeling and Next Sentence Prediction. In MLM, random words in a sentence are masked, and the model learns to predict these words based on their context. NSP involves determining whether a sentence logically follows another. This extensive pre-training enables BERT to develop a deep understanding of language patterns and structures.

Once pre-trained, BERT can be fine-tuned for specific NLP tasks with additional output layers. This fine-tuning adapts the model to particular requirements or datasets.

In this project, we used the R package `text` to interface with Python's Transformers package [Kjell et al., 2023]. This setup required Python3 and its key statistical and mathematical libraries to be installed on our machine. Additionally, we employed the `reticulate` package in R. This package allows for running Python code within R, enabling us to access Python objects and convert data between Python and R [Kalinowski et al., 2023].

Our approach involved using the "bert-base-uncased" model from the pre-trained BERT series, applied through the `text::textEmbed` function. Processing all 50 comment threads serially initially

took over four hours. The resulting file of text embeddings was 7.8 GB. To speed up this process, we used the R packages `future` and `future.apply`, which allowed us to parallelize the task. We divided the 50 comment threads into 10 groups, each with 4 threads, and processed them simultaneously across seven available cores of our machine. This parallel processing was implemented using the `process_threads_parallel function`, described in section 9.2. This function was developed to parallelize the `text::textEmbed` function effectively. By adopting this approach, we managed to reduce the total processing time to under 45 minutes.

The processing was carried out on a 2021 Apple MacBook Air with 16GB RAM and an M1 processor with 8 cores.

5.2 Dimension Reduction

Principal Component Analysis (PCA) is a statistical technique used for dimensionality reduction. It works by identifying the most significant features of the data, which capture the majority of the variance in the dataset. PCA transforms the original data into a new set of variables, known as principal components. These components are orthogonal and the first (relative to the number of components) few capture most of the variation present in the original dataset. This reduction in dimensions is achieved by projecting the original data onto a smaller subspace while retaining the essence of the original data as much as possible.

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a machine learning algorithm used for non-linear dimensionality reduction. It is suitable for the visualization of high-dimensional datasets. t-SNE works by converting similarities between data points into joint probabilities and then minimizing the Kullback–Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data.

Combining PCA and t-SNE is an approach that reduces the dimensions of text embeddings before clustering. PCA is used first to reduce the dimensionality to a manageable level while preserving as much of the variance as possible. t-SNE, while effective at capturing the local structure of data, can be computationally expensive and less effective in very high-dimensional spaces. By reducing dimensions initially with PCA, t-SNE can then operate more efficiently and effectively.

Universal Manifold Approximation and Projection (UMAP) is a technique for dimension reduction, capable in handling large, high-dimensional datasets [McInnes and Healy, 2018]. UMAP is built in the mathematical framework of Riemann geometry and algebraic topology called Fuzzy Topology. It works by constructing a high-dimensional graph representation of the data and then optimally laying out this graph in a lower-dimensional space. This process involves first approximating the manifold structure of the data by finding the nearest neighbors for each point and then optimizing the embedding so that points that were close in the high-dimensional space remain close in the reduced space.

UMAP offers several advantages over PCA and t-SNE [Oskolov, 2021]. Unlike PCA, which is a linear technique and may not capture the non-linear relationships in the data, UMAP can handle non-linearities effectively. Compared to t-SNE, UMAP generally preserves more of the global structure of the data and is faster, making it more suitable for large datasets. Additionally, UMAP is often more robust in preserving the local and global structure of the data. However, UMAP also has some drawbacks as it can sometimes be sensitive to the choice of parameters, such as the number of nearest neighbors or the minimum distance between points in the low-dimensional

representation. t-SNE is known for its ability to reveal clusters at many scales where as UMAP can sometimes merge distinct clusters, especially when the differences between them are subtle.

We utilized the R package `Rtsne`, which is a wrapper around the fast T-distributed Stochastic Neighbor Embedding implementation by Van der Maaten, written in C++ [Krijthe, 2023]. We used the R package `umap`, using the first of its two implementations, being the written for R version, for embedding and not the wrapper for the python package `umap-learn` [Konopka, 2023]. The embeddings were centered and scaled for PCA. t-SNE has notable tuning parameter called perplexity, which says how to balance the local and global aspects of the data by telling the algorithm how many close neighbors each point has. In the Clustering section below we discuss this optimization. It is worth noting that the author noticed UMAP finished running much faster than PCA with t-SNE for the same embedding matrix for both GMM and K-means clustering. We did not record computation times.

5.3 Clustering

K-means clustering is a unsupervised learning algorithm used to partition datasets into a pre-specified number of clusters k. It operates by iteratively refining the positions of centroids, which are the representatives of the clusters. At first, k centroids are randomly placed and data points are assigned to the nearest centroid based on a distance metric, commonly Euclidean distance. Subsequent iterations recalibrate these centroids by calculating the mean of all points assigned to each cluster. The process repeats until centroid positions stabilize, signaling the emergence of distinct groupings.

Gaussian Mixture Models adopt a probabilistic model-based approach to clustering. They presume that the data points are generated from a finite mixture of Gaussian distributions, each characterized by its own mean and covariance. GMM's try estimate these parameters, thereby uncovering the latent Gaussian distributions that govern the data structure. Unlike K-means, GMM's accommodates what is called soft clustering, providing a probability-based membership of each data point to the clusters. This makes GMMs capable of handling clusters of varying sizes and different covariance structures.

In Section 5.3, the concept of modular code is exemplified. One of the primary objectives was to develop scalable code, essential for large scale data analysis projects. The function `combine_embeddings` aggregates the embedding matrices from grouped threads, creating a comprehensive embedding matrix for each set of five threads.

The `clean_comments` function mirrors the preprocessing steps delineated in Section 4.1, ensuring consistency in data treatment throughout the study. Complementing this, `concatenate_comments` efficiently compiles and chronologically orders the textual content from each comment thread, aligning it with the corresponding embeddings.

Central to the optimization process are the `perform_clustering` and `calculate_silhouette_score` functions. These auxiliary functions execute specific clustering algorithms and evaluate the clustering quality using the mean silhouette score, respectively. They are called in the `perform_optimization` function, which searches across various configurations. This function iteratively explores combinations of dimension reduction and clustering methods, varying the number of clusters (ranging from 2 to 5) and perplexity values (specifically 5, 15, and 30 for t-SNE). It then calculates the mean silhouette score for each configuration, guiding the identification of

the most effective clustering approach.

The `combine_cluster_comments` function integrates the clustering labels obtained from the embedding matrices with their corresponding textual and metric data.

`process_and_cluster_comments` takes a pair of dimension reduction and clustering methods, performs optimization, and consolidates the results into a final data frame. This frame includes not only the processed text and metrics but also the optimal number of clusters, silhouette score, and, if applicable, the optimal perplexity value.

Finally, `process_and_save_comments` extends this modular framework across all four method pairs. For each set of embeddings, it selects the best configuration based on the mean silhouette score. The result is a collection of ten data frames, each encapsulating the original comment data, the optimal cluster configuration, and the associated parameters derived from the optimization process. This comprehensive approach exemplifies the project's scalability and robustness.

5.4 Linear Discriminant Analysis

LDA is a statistical model used in Natural Language Processing for the purpose of topic analysis. LDA makes the assumption that documents are a mixture of topics, where each topic is characterized by a distribution of words. This model is adept at uncovering underlying thematic structures in large collections of text data.

In the context of this project, LDA is implemented through the integration of the `topicmodels` and `tidytext` packages in R. The process begins with the construction of a Document-Term Matrix (DTM). The DTM encapsulates the frequency of words across different documents, where each row represents a document (in this case, a comment) and each column signifies a word. The formation of the DTM involves the removal of common stopwords that are typically uninformative for topic analysis such as "and", "their,"or" and "so".

Once the DTM is established, the LDA model is applied. In this analysis, the number of topics (denoted as num_topics = 5) is predetermined, and the LDA algorithm iteratively learns the word distributions for each topic and the topic distribution for each document. This iterative process is guided by a probabilistic approach that assigns words to topics based on their co-occurrence patterns and the current state of topic assignments [Grün and Hornik, 2023].

The output of the LDA model includes two key components: the top terms in each topic, which provide insights into the thematic essence of each topic, and the dominant topic for each document, offering a perspective on the primary theme present in each comment. The top terms are identified based on their beta values, a measure of the word's significance within a topic. In parallel, the dominant topic of each document is determined based on the gamma values, reflecting the probability of the document belonging to a particular topic.

We integrate the LDA results with the original data to merge the dominant topic back to each comment. This process reveals the latent topics within the comments and aligns each comment with its most prominent thematic element.

5.5 Sentiment Analysis

Sentiment analysis refers to the computational technique of identifying and categorizing opinions expressed in a piece of text, especially in order to determine whether the writer's attitude is positive, negative, or neutral. This analysis enables the understanding of sentiments embedded in large volumes of text data [Hutto and Gilbert, 2014].

The R package `vaderSentiment`, a wrapper for the Python package `VADER` (Valence Aware Dictionary and sEntiment Reasoner), is a lexicon and rule-based sentiment analysis tool specifically attuned to sentiments expressed in social media. VADER uses a combination of a sentiment lexicon, which is a list of lexical features (e.g., words) that are generally labeled according to their semantic orientation as either positive or negative. The package evaluates text, including slang, to provide a sentiment score that ranges from -1 (most negative) to 1 (most positive).

In this project, `vaderSentiment` is applied to analyze the sentiment of Reddit comments, facilitated by the `reticulate` package in R.

6 Results

The clustering optimization results, as depicted in Table 1, reveal a preference for PCA combined with t-SNE in seven out of ten sets of text embeddings. This outcome was somewhat unexpected, given UMAP's efficiency in handling overlapping clusters typically associated with textual data. The preference for PCA and t-SNE may be attributed to the limitations of the R implementation of UMAP, which, unlike its Python counterpart, lacks tunable parameters for fine-tuning. Consequently, the simpler method was favored in the absence of detailed control mechanisms.

K-means clustering emerged as the preferred method in eight out of the ten embedding sets. The study did not explore other models beyond Gaussian Mixture Models, such as those available in `mclust` package. The possibility remains that alternative mixture models might have better fitted the data with appropriate tuning. The predominance of the K-means clustering algorithm suggests that, in certain scenarios, simpler methods may outperform more complex ones, especially when the latter lack nuanced control.

The number of clusters ranged from 2 to 3, aligning with sociological expectations of categorizing sentiments into positive, negative, and neutral perspectives regarding the conflict. This finding reinforces the notion that complexity in methods is advantageous primarily when accompanied by precise control, echoing the adage "simplicity is best."

Figure 3 presents the distribution of sentiment scores across clusters for each set of embeddings. Notably, all mean sentiment scores per cluster are negative, reflecting the contentious and distressing nature of the Israel-Palestine conflict. The typical pattern observed includes one cluster with mostly neutral sentiments (scores > 0.10) and at least one negative cluster (scores < -0.20).

Figure 4 illustrates the sentiment trends over time, aggregating all comments, arranging them chronologically, and averaging sentiment scores on a weekly basis. This temporal sentiment analysis reveals significant fluctuations, necessitating further examination to understand the underlying causes. Correlating these sentiment spikes with major news events could elucidate the reasons behind these changes. The overall trend does not clearly indicate a decreasing public sentiment toward the conflict, highlighting the need for more in-depth analysis.

Figures 1 and 2 showcase the sentiment distribution across topics and clusters for each set of embeddings, respectively. These figures also illustrate the range of time periods for each set of chronologically ordered embeddings. The choice of five topics for LDA was somewhat arbitrary, based on the presumption that it would be a reasonable number for a global conflict of this magnitude. However, a more rigorous statistical approach to determine the optimal number of topics would be beneficial.

An examination of the top words in each topic revealed a dominance of commonly recurring terms such as "Hamas," "Israel," "Gaza," and "war." To provide a more nuanced understanding of the topics, future analyses could consider excluding these predominant terms to see if the remaining top terms offer a clearer distinction among topics. Such an approach might yield more informative insights into the thematic variations within the discourse.

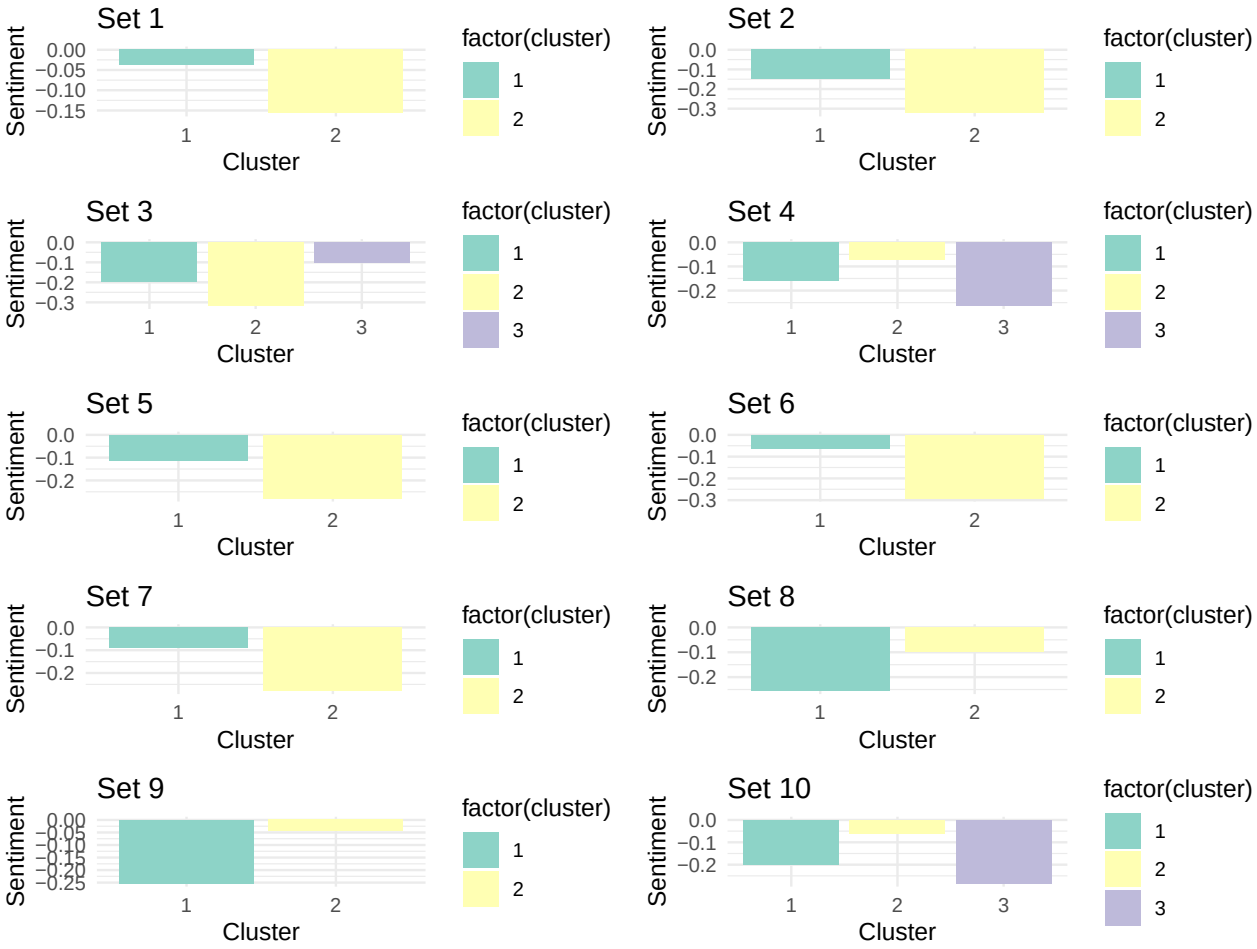

Figure 3: Average Sentiment across all 10 sets of embeddings within each cluster. The Set numbers can be matched chronollogically to the date titles in Figures 1 and 2

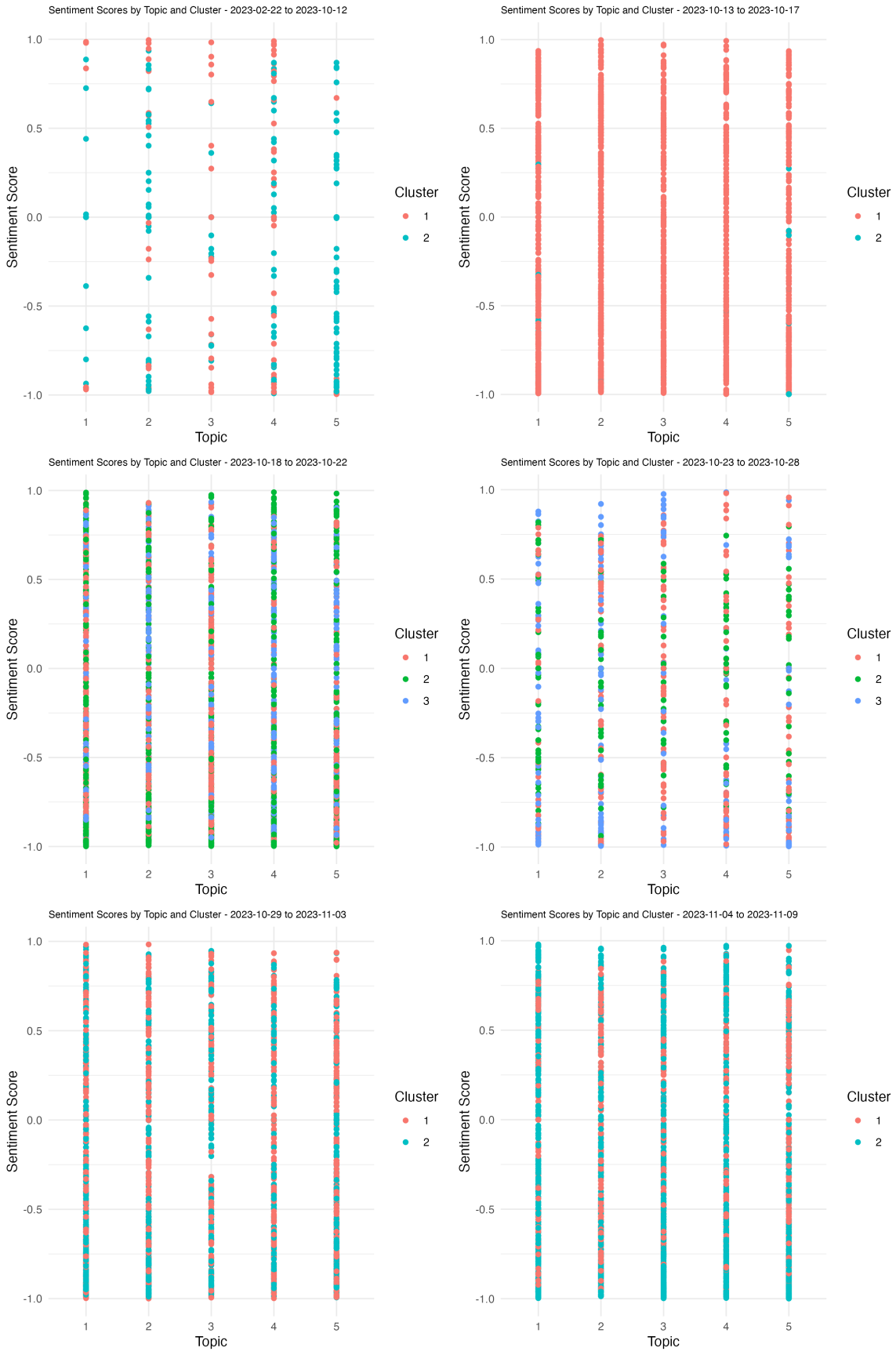

Figure 1: Distribution of sentiment scores and clusters among topics for the described dates.

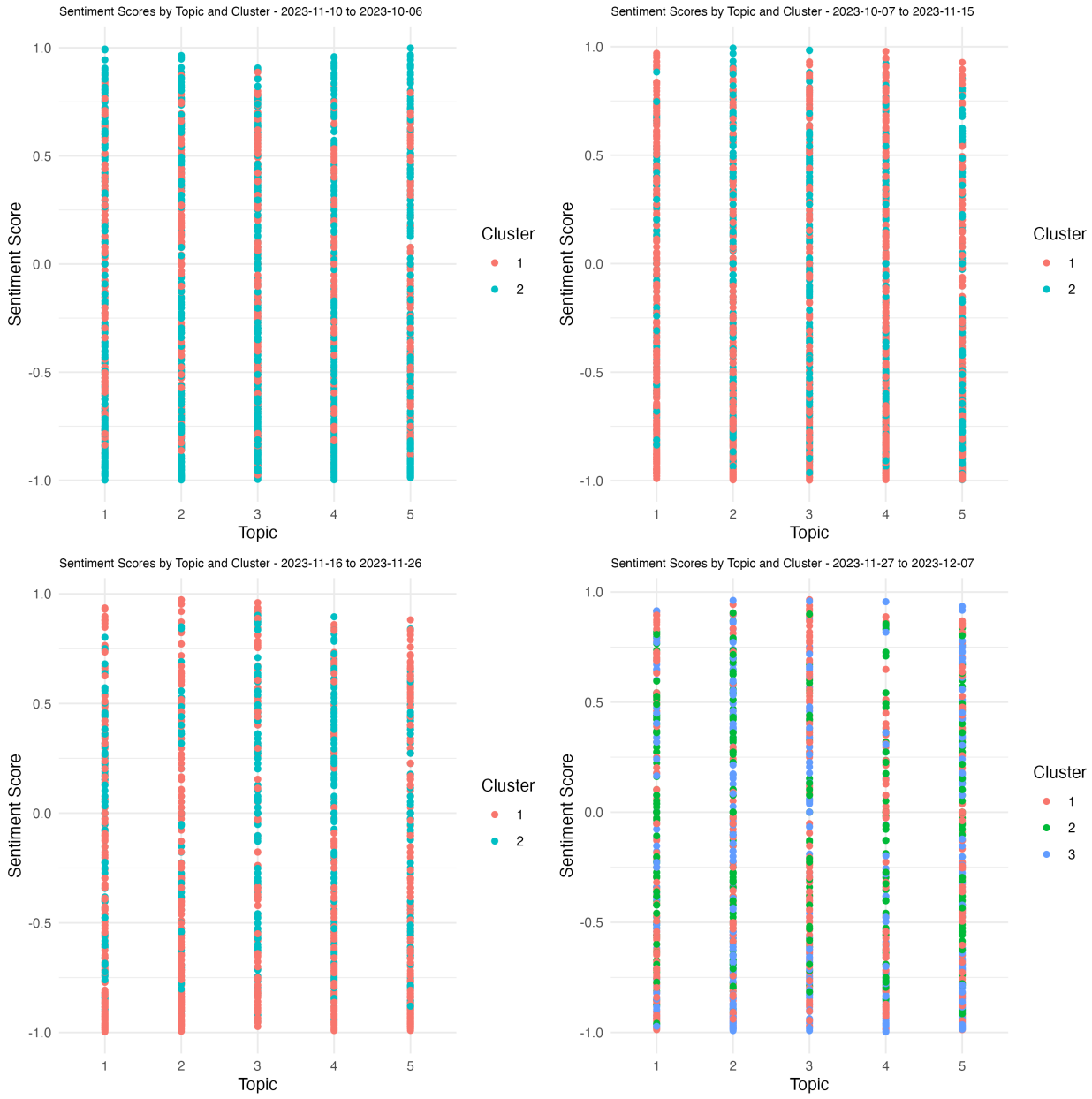

Figure 2: Distribution of sentiment scores and clusters among topics for the described dates.

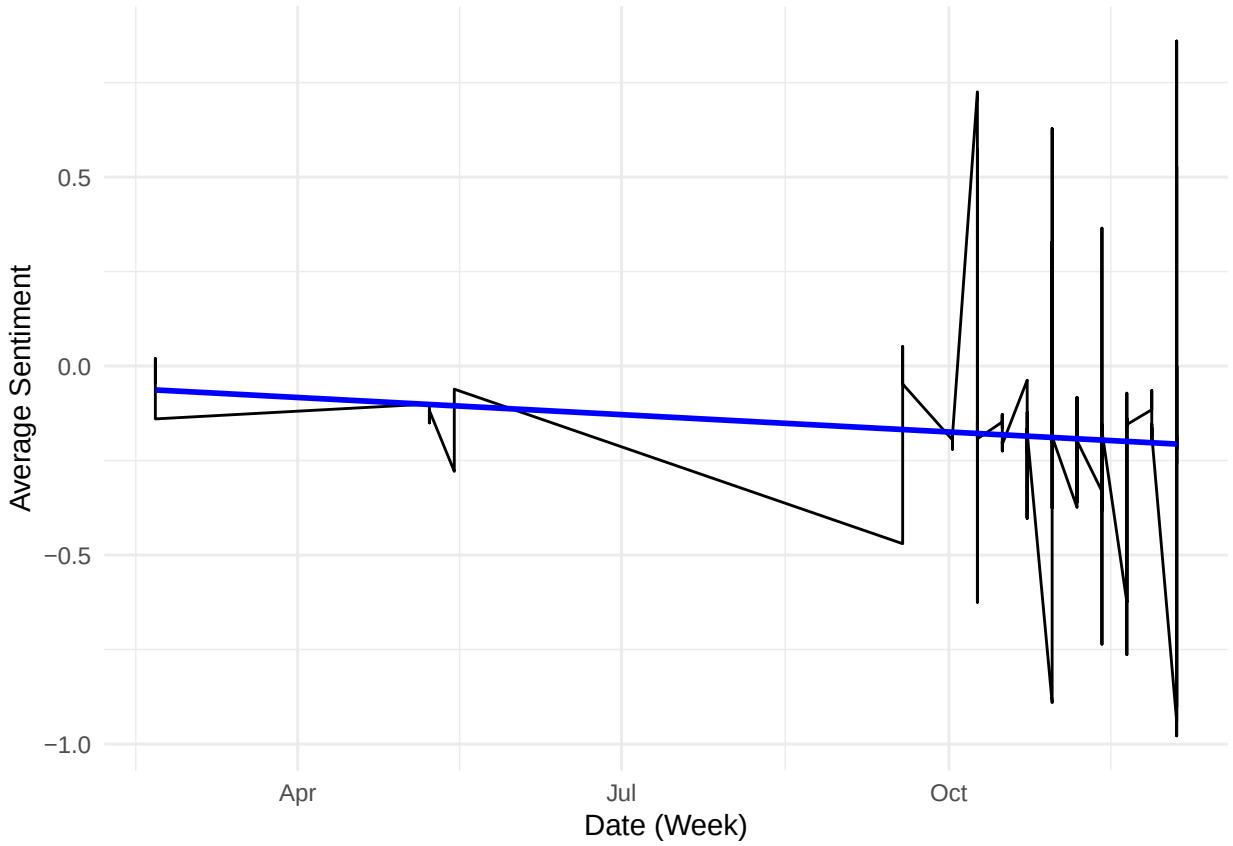

Figure 4: Sentiment score using vaderSentiment averaged over all comments by week and fit with simple linear regression trend line.

7 Discussion

This project highlights the need for a detailed review of the current methods used in clustering and dimension reduction for analyzing social media comments. It's clear that refining these methods can lead to better results. For instance, the `mclust` package allows for more precise control over

Table 1: Clustering Analysis Summary

Embedding_Number	Reduction_Method	Clustering_Method	Number_of_Clusters	Perplexity	Mean_Silhouette_Score
1	UMAP	kmeans	2	NA	0.4828342
2	UMAP	kmeans	2	NA	0.7174337
3	PCA	GMM	3	15	0.3947601
4	UMAP	kmeans	3	NA	0.4134959
5	PCA	kmeans	2	15	0.4128419
6	PCA	GMM	2	30	0.4252567
7	PCA	kmeans	2	15	0.4109411
8	PCA	kmeans	2	30	0.4255994
9	PCA	kmeans	2	30	0.4066477
10	PCA	kmeans	3	30	0.4060226

how clusters are formed through the specification of the variance structure, which is crucial given the complex nature of text data. By understanding the detailed structures in text embeddings, we can improve how we group these data.

In terms of dimension reduction, exploring tools like the Python version of `umapLearn` could provide deeper insights. This tool, specifically adjusted for text, might reveal more meaningful patterns hidden in the high-dimensional space of written language.

A major challenge we faced was moving beyond a standard pre-trained BERT model. Developing a customized model, trained with specific labeled data, would likely improve our analysis. This requires setting up a system to continuously collect data from Reddit, using technologies like Kafka and an SQL database. This setup would help train the model to recognize political biases in the comments, focusing on differentiating left and right-wing viewpoints.

Originally, we wanted to analyze how different groups view the Israel-Palestine conflict by classifying Reddit subreddits on a political spectrum. We planned to use a specialized model to determine the general sentiment in these subreddits over time. This approach aimed to use Reddit as a sample to understand broader perceptions of the conflict across different political groups. However, this task turned out to be very complex and time-consuming. We had to scale down our ambitions to analyzing comments from one subreddit over a shorter period. This smaller-scale study still proved to be a challenging task, reflecting the complexity of working with social media data and the need for advanced, specific methods to fully understand it. In the future, we aim to develop better tools and approaches to tackle these challenges more effectively.

8 Conclusion

This project explored the sentiments and themes present in online discourse surrounding the Israel-Palestine conflict, as reflected in Reddit comments. Employing a blend of NLP techniques, including BERT embeddings, dimensionality reduction (via PCA and UMAP), and clustering algorithms (K-means and Gaussian Mixture Models), the project aimed to distill and analyze the nuanced spectrum of public opinion and discussion topics over a defined time period.

A predominant finding of the study is the overall negative sentiment in the discourse relecting the contentious and emotive nature of the subject matter. The analysis revealed that simpler clustering methods, such as K-means, often surpassed more intricate models in effectiveness. This outcome suggests a notable efficiency in straightforward analytical methods in certain contexts, especially when paired with text data. However, the study faced limitations in the contextual analysis of the relationships between clustering, topis and sentiment.

While this project has ever so lightly shed light on the dynamics of online discussions regarding the Israel-Palestine conflict, it also opens the door to further analysis by using the robust modular code base we developed. By addressing current limitations and expanding the methodological framework, future research can provide deeper, more actionable insights into global public sentiment and discourse over any current event of interest.

References

Reddit API. https://www.reddit.com/dev/api, 2023. Accessed: date-of-access.

Ingo Feinerer and Kurt Hornik. *Text Mining Package 'tm'*, 2023. URL https://cran.r-project.org/web/packages/tm/tm.pdf.

Greg Forkutza. Final Project Repository. https://github.com/GregForkutza/UMAP, 2023.

Bettina Grün and Kurt Hornik. *Topicmodels: An R Package for Fitting Topic Models*, 2023. URL https://cran.r-project.org/web/packages/topicmodels/vignettes/topicmodels.pdf.

Clayton Hutto and Eric Gilbert. VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 8, pages 216–225, 2014. doi: 10.1609/icwsm.v8i1.14550. URL https://doi.org/10.1609/icwsm.v8i1.14550.

Tomasz Kalinowski, Kevin Ushey, JJ Allaire, and Yuan Tang. *reticulate: Interface to 'Python'*, 2023. URL https://cran.r-project.org/web/packages/reticulate/index.html.

Oscar Kjell, Salvatore Giorgi, and Andrew Schwartz. *text: Analyses of Text using Transformers Models from HuggingFace, Natural Language Processing and Machine Learning*, 2023. URL https://cran.r-project.org/web/packages/text/index.html.

Tomasz Konopka. map: Uniform manifold approximation and projection, 2023. URL https://cran.r-project.org/web/packages/umap/index.html.

Jesse Krijthe. Rtsne: T-distributed stochastic neighbor embedding using a barnes-hut implementation, 2023. URL https://cran.r-project.org/web/packages/Rtsne/index.html.

Leland McInnes and John Healy. Umap: Uniform manifold approximation and projection for dimension reduction. *ArXiv e-prints*, 1802.03426, 2018. URL https://arxiv.org/abs/1802.03426.

Nikolay Oskolov. t-SNE vs UMAP: Global Structure, 2021. URL https://towardsdatascience.com/tsne-vs-umap-global-structure-4d8045acba17. Accessed: 2023-12-01.

Hadley Wickham. *httr: Tools for Working with URLs and HTTP*, 2022. URL https://CRAN.R-project.org/package=httr.

9 Supplementary Material

9.1 Reddit API Wrapper

```
## Reddit API Wrapper

  # Connect to Reddit API using httr
    # Define API credentials
    client_id <- "insert here"
    client_secret <- "insert here"

    reddit_endpoints <- oauth_endpoint(
      authorize = "https://www.reddit.com/api/v1/authorize",
      access = "https://www.reddit.com/api/v1/access_token"
    )

    # Authenticate
    my_app <- oauth_app("reddit", key = client_id, secret = client_secret)

    reddit_token <- oauth2.0_token(
      reddit_endpoints,
      my_app,
      scope = c("read"),
      use_basic_auth = TRUE,
      config_init = user_agent("Mac:UMAP:v1.0 (by /u/Forkman3939)")
    )

  # Function to retrieve urls of JSON request
  get_url <- function(subreddits, keywords, time_frame, sort_parm, limit) {
    # Construct the search query
    query <- paste0("subreddit:", subreddits[1], "+AND+(",
                    paste(keywords, collapse=" OR "), ")")

    # Initialize a vector to store post URLs
    permalinks <- c()

    # Loop through each subreddit
    for(subreddit in subreddits) {
      search_url <- paste0(
        "https://oauth.reddit.com/r/", subreddits[1], "/search.json?q=",
        URLencode(query), "&sort=", sort_parm, "&t=",
        time_frame, "&limit=", limit)

      # Send the request
      response <- GET(search_url, add_headers(
        Authorization = paste("Bearer", reddit_token$credentials$access_token)))
```

```
    # Check if the request was successful
    if (http_status(response)$category == "Success") {
      # Parse the response
      content_response <- content(response, as = "text")
      content_list <- fromJSON(content_response, flatten = TRUE)

      # Extract URLs from content_list
      for (i in 1:nrow(content_list$data$children)) {
        # Extract the permalink
        permalink <- content_list$data$children[i, "data.permalink"]
        # Append the new permalink to the existing vector
        permalinks <- c(permalinks, permalink)
      }
    }
  }

  # Modify URLs to the correct format
  corrected_urls <- lapply(permalinks, function(link) {
    # Construct the full URL and add it to the list
    post_urls <- paste0("https://oauth.reddit.com", link)

  })

  # Flatten the list to a vector
  corrected_urls <- unlist(corrected_urls)
  # Return a list containing both sets of URLs
  return(list(permalinks = permalinks, corrected_urls = corrected_urls))

}

# Define function to retrieve comments from JSON query and extract as a nested
# list structure
get_content <- function(url_vec) {
  result <- list()
  for (i in 1:length(url_vec)) {
    # Fetch the comments using the GET request
    response <- GET(url_vec[i], add_headers(
      Authorization = paste("Bearer", reddit_token$credentials$access_token)))
    #Extract the content from the response
    content_response <- content(response)
    # Extract post info from the content
    post_list <- content_response[[1]]
    # Extract the comment info from the content
    comments_list <- content_response[[2]]
    # Store post and comment info
    result[[i]] <- list(post_list = post_list, comments_list = comments_list)
  }
```

```
  return(result)
}

# Define function to recursively extract features from nested list structure
get_comments <- function(comments, parent_id = "ROOT", depth = 0) {
  results <- data.frame(
    ups = integer(),
    downs = integer(),
    comment_id = character(),
    author = character(),
    parent_id = character(),
    subreddit = character(),
    link_id = character(),
    score = integer(),
    is_submitter = logical(),
    body = character(),
    depth = integer(),
    controversiality = integer(),
    created_utc = integer(),
    stringsAsFactors = FALSE
  )
  print(paste(
    "Processing depth:", depth, "with",
    length(comments$data$children), "comments"))

  for (i in 1:length(comments$data$children)) {
    comment <- comments$data$children[[i]]$data

    # Check for NULL values and replace with default values if necessary
    ups <- ifelse(is.null(comment$ups), 0, comment$ups)
    downs <- ifelse(is.null(comment$downs), 0, comment$downs)
    comment_id <- ifelse(is.null(comment$id), NA, comment$id)
    author <- ifelse(is.null(comment$author), NA, comment$author)
    parent_id <- ifelse(is.null(comment$parent_id), NA, comment$parent_id)
    subreddit <- ifelse(
      is.null(comment$subreddit_name_prefixed), NA,
      comment$subreddit_name_prefixed)
    link_id <- ifelse(is.null(comment$link_id), NA, comment$link_id)
    score <- ifelse(is.null(comment$score), 0, comment$score)
    is_submitter <- ifelse(is.null(comment$is_submitter), FALSE,
                           comment$is_submitter)
    body <- ifelse(is.null(comment$body), "", comment$body)
    depth <- ifelse(is.null(comment$depth), 0, comment$depth)
    controversiality <- ifelse(is.null(comment$controversiality), 0,
                               comment$controversiality)
    created_utc <- ifelse(is.null(comment$created_utc), 0,
                          comment$created_utc)
```

```
    results <- rbind(results, data.frame(
      ups = ups,
      downs = downs,
      comment_id = comment_id,
      author = author,
      parent_id = parent_id,
      subreddit = subreddit,
      link_id = link_id,
      score = score,
      is_submitter = is_submitter,
      body = body,
      depth = depth,
      controversiality = controversiality,
      created_utc = created_utc,
      stringsAsFactors = FALSE
    ))

    # Check if replies exist and contain actual data
    if (!is.null(comment$replies) && is.list(comment$replies) &&
        !is.null(comment$replies$data) &&
        length(comment$replies$data$children) > 0) {
      print(paste(
        "Found replies at depth:", depth, "for comment ID:", comment$id))
      replies_results <- extract_comments(
        comment$replies, comment$id, depth +1)

      # Only bind if replies_results has rows
      if (nrow(replies_results) > 0) {
        results <- rbind(results, replies_results)
      }
    }
  }

  return(results)
}

# Wrapper for functions to take JSON query and return a list of data frames
# containing comment text and other variables
get_data <- function(subreddits, keywords, time_frame, sort_parm, limit) {

  urls_vec <- get_url(subreddits, keywords, time_frame, sort_parm, limit)
  content <- get_content(urls_vec)
  df <- list()

  for(i in 1:(length(content))) {
    df[[i]] <- get_comments(content[[2]])
  }
```

```
  return(df)
}
```

9.2 BERT Transformer using Parallel Processing

```
## BERT Transformer using parallel processing

  # Function to process comments and get embedding
  process_comments <- function(df) {
    df_clean <- df %>%
      mutate(clean_body = gsub("[^[:alnum:] ]", "", text)) %>%
      mutate(clean_body = tolower(clean_body)) %>%
      filter(text != "[deleted]", text != "[removed]")

    comments <- df_clean$clean_body
    comments <- comments[comments != "" & !is.na(comments)]

    if (length(comments) > 0) {
      textEmbed(
        texts = comments,
        model = "bert-base-uncased",
        layers = -2,
        aggregation_from_layers_to_tokens = "mean",
        aggregation_from_tokens_to_texts = "mean",
        keep_token_embeddings = TRUE
      )
    } else {
      NULL
    }
  }

  # Function to save embedding
  save_embeddings <- function(embeddings, file_name) {
    saveRDS(embeddings, file = file_name)
  }

  # Function to generate file name
  generate_filename <- function(base_name, index) {
    parts <- unlist(strsplit(base_name, "_"))
    paste0("data/embeddings/", paste(parts, collapse = "_"), "_", index, ".rds")
  }

  # Process all threads in parallel with progress tracking
  process_threads_parallel <- function(df_list, base_name) {
    num_cores <- detectCores() - 1
    plan(multisession, workers = num_cores)
```

```
  # Split df_list into batches of 5
  batches <- split(df_list, ceiling(seq_along(df_list) / 5))

  # Process each batch in parallel
  future_lapply(seq_along(batches), function(i) {
    print(paste("Processing batch", i, "of", length(batches)))
    batch <- batches[[i]]
    all_embeddings <- lapply(batch, function(df_item, index) {
      print(paste("  - Processing thread", index, "in batch", i))
      df <- df_item$nodes %>% as.data.frame()
      result <- process_comments(df)
      print(paste("  - Finished thread", index, "in batch", i))
      result
    }, index = seq_along(batch))
    all_embeddings <- all_embeddings[!sapply(all_embeddings, is.null)]

    if (length(all_embeddings) > 0) {
      save_embeddings(all_embeddings, generate_filename(base_name, i))
      print(paste("Saved embeddings for batch", i))
    }
  })
}

# Load data and process in parallel with progress tracking
df_list <- readRDS(file = "news_top_year_50_comments.rds")
base_name <- gsub("\\.rds$", "", basename("news_top_year_50_comments.rds"))
process_threads_parallel(df_list, base_name)
```

9.3 Data Processing: Pre-processing, Dimension Reduction and Clustering

```
# Define function to combine embedding matrices for each thread
# row wise for list of more than one embedding.
combine_embeddings <- function(all_embeddings) {

  # Initialize an empty matrix to store all embedding
  combined_embeddings_matrix <- NULL
  # Loop through each thread's embedding and concatenate them
  for (embeddings in all_embeddings) {
    # Extract the embedding matrix for the current thread
    embeddings_matrix <- do.call(rbind, embeddings$embedding$texts$texts) %>%
      t()
    print(dim(embeddings_matrix))
    # Combine with the main embedding matrix
    if (is.null(combined_embeddings_matrix)) {
      combined_embeddings_matrix <- embeddings_matrix
```

```
      } else {
        combined_embeddings_matrix <- rbind(
          combined_embeddings_matrix, embeddings_matrix)
      }
    }
    return(combined_embeddings_matrix)
  }

# Define function to concatenate comment thread for list with multiple embedding

  # Define function to clean comments
  clean_comments <- function(df) {
    df_clean <- df %>%
      mutate(clean_body = gsub("[^[:alnum:] ]", "", text)) %>%
      mutate(clean_body = tolower(clean_body)) %>%
      filter(text != "[deleted]", text != "[removed]") %>%
      select(clean_body, score, children, date)
    df_clean <- na.omit(df_clean)
    print(dim(df_clean))
    return(df_clean)
  }

  # Define function to loop over df in list
  concatenate_comments <- function(dfs, size_lower, size_upper) {
    # Initialize an empty data frame to store all comments and metrics
    all_comments_df <- data.frame()
    for (i in size_lower:size_upper) {
      clean_df <- clean_comments(dfs[[i]]$nodes)
      all_comments_df <- rbind(all_comments_df, clean_df)
    }
    return(all_comments_df)
  }

# Define function to reduce dimensions and cluster optimizing over
# perplexity and number of clusters. Chooses best model over 2x2 methods
# using silhouette score ranking.
perform_optimization <- function(embedding_matrix, dim_reduction_method,
                                 cluster_method, cluster_range = 2:5) {
  set.seed(123)

  # Define specific perplexity values for PCA+t-SNE
  perplexity_range <- c(5, 15, 30)
  # Initialize variables to store the best configuration
  best_silhouette_score <- 0
  optimal_n_clusters <- 2
  optimal_perplexity <- 5
  # Dimension Reduction and Clustering
```

```
  if (dim_reduction_method == "pca") {
    pca_result <- prcomp(embedding_matrix, center = TRUE, scale. = TRUE)
    for (perplexity in perplexity_range) {
      tsne_result <- Rtsne(
        pca_result$x[, 1:min(ncol(pca_result$x), 50)], dims = 2,
        perplexity = perplexity, check_duplicates = FALSE)
      results <- tsne_result$Y
      # Find optimal number of clusters for this perplexity
      for (n in cluster_range) {
        clustering_result <- perform_clustering(results, cluster_method, n)
        silhouette_score <- calculate_silhouette_score(
          clustering_result, results)

        if (silhouette_score > best_silhouette_score) {
          best_silhouette_score <- silhouette_score
          optimal_n_clusters <- n
          optimal_perplexity <- perplexity
        }
      }
    }
    # Final clustering with best perplexity and number of clusters
    final_tsne_result <- Rtsne(
      pca_result$x[, 1:min(ncol(pca_result$x), 50)], dims = 2,
      perplexity = optimal_perplexity, check_duplicates = FALSE)
    final_results <- final_tsne_result$Y
  } else if (dim_reduction_method == "umap") {
    umap_result <- umap(embedding_matrix)
    final_results <- umap_result$layout
    # Optimize only the number of clusters for UMAP
    for (n in cluster_range) {
      clustering_result <- perform_clustering(final_results, cluster_method, n)
      silhouette_score <- calculate_silhouette_score(clustering_result,
                                                     final_results)

      if (silhouette_score > best_silhouette_score) {
        best_silhouette_score <- silhouette_score
        optimal_n_clusters <- n
      }
    }
  } else {
    stop("Invalid value for dimension reduction method")
  }
  # Perform final clustering with optimal parameters
  final_clusters <- perform_clustering(final_results,
                                       cluster_method, optimal_n_clusters)
  return(list(
    embeddings = final_results,
```

```
    cluster = final_clusters,
    optimal_n_clusters = optimal_n_clusters,
    best_silhouette_score = best_silhouette_score,
    optimal_perplexity = ifelse(
      dim_reduction_method == "pca", optimal_perplexity, NA)
  ))
}

  # Helper function for clustering
  perform_clustering <- function(data, method, n_clusters) {
    if (method == "kmeans") {
      return(kmeans(data, centers = n_clusters)$cluster)
    } else if (method == "gmm") {
      return(Mclust(data, G = n_clusters)$classification)
    } else {
      stop("Invalid clustering method")
    }
  }

  # Helper function to calculate silhouette score
  calculate_silhouette_score <- function(clustering_result, data) {
    dist_mat <- dist(data)
    return(mean(silhouette(clustering_result, dist_mat)[, "sil_width"]))
  }

# Function to combine comments with their corresponding cluster
combine_cluster_comments <- function (comments, embeddings_matrix, clusters) {
  df_final <- data.frame()
  if (length(comments) == nrow(embeddings_matrix)) {
    df_final <- data.frame(comment = comments, cluster = clusters)
  } else {
    warning(
 "Mismatch between number of comments and number of rows in embeddings matrix")
  }
  df_final <- df_final[sample(nrow(df_final)),]
  return(df_final)
}

# Wrapper function for all other processing, cleaning and helper functions.
process_and_cluster_comments <- function(
    all_embeddings, dfs, size_lower, size_upper,
    dim_reduction_method, cluster_method) {
  combined_embeddings_matrix <- combine_embeddings(all_embeddings)
  all_comments_df <- concatenate_comments(dfs, size_lower, size_upper)

  # Perform optimization to get the best clustering configuration
  cluster_results <- perform_optimization(
```

```
    combined_embeddings_matrix, dim_reduction_method, cluster_method)

  # Combine the comments and metrics with clusters
  final_df <- combine_cluster_comments(
    all_comments_df$clean_body, combined_embeddings_matrix,
    cluster_results$cluster)

  # Add the metrics and optimal configuration to the final data frame
  final_df <- cbind(final_df, all_comments_df[, c("score", "children", "date")],
                    optimal_n_clusters = cluster_results$optimal_n_clusters,
                    best_silhouette_score = cluster_results$best_silhouette_score,
                    optimal_perplexity = cluster_results$optimal_perplexity)

  return(final_df)
}

# Loop over all embedding files and pairs of dimension reduction and
# clustering methods.

  # Function to determine lower and upper bounds based on file name
  get_bounds <- function(file_name) {
    # Extract the number right before ".rds"
    num <- as.numeric(sub(".*_([0-9]+)\\.rds$", "\\1", basename(file_name)))

    lower <- num
    upper <- ifelse(num == 1, 3, ifelse(num == 4, 10, num + 4))
    return(c(lower, upper))
  }

# Iterates over all methods and parameters and chooses the best model according
#  to silhouette score and saves as unique .rds file in chosen directory
process_and_save_comments <- function(
file_names, dfs, reduction_methods, clustering_methods) {
  for (file_path in file_names) {
    all_embeddings <- readRDS(file_path)
    bounds <- get_bounds(file_path)
    lower <- bounds[1]
    upper <- bounds[2]
    best_overall_score <- -Inf
    best_df <- NULL
    best_method_combination <- NULL

    for (reduction_method in reduction_methods) {
      for (clustering_method in clustering_methods) {
        result_df <- process_and_cluster_comments(
          all_embeddings, dfs, lower, upper, reduction_method,
          clustering_method)
```

```
        # Check if this combination has the best silhouette score
        current_best_score <- max(result_df$best_silhouette_score, na.rm = TRUE)
        if (current_best_score > best_overall_score) {
          best_overall_score <- current_best_score
          best_df <- result_df
          best_method_combination <- paste(
            reduction_method, clustering_method, sep = "_")
        }
      }
    }
    # Save the best dataframe
    if (!is.null(best_df)) {
      df_name <- paste0(
        "final_df_", lower, "_", upper, "_", best_method_combination)
      saveRDS(best_df, paste0("data/final_df/", df_name, ".rds"))
    }
  }
}

# Function to extract the numeric part from the file name
extract_number <- function(file_name) {
  as.numeric(sub(".*_([0-9]+)\\.rds$", "\\1", basename(file_name)))
}

# Execute the function
file_names <- list.files(
  path = "data/embeddings", full.names = TRUE, pattern = "\\.rds$")
file_names <- file_names[order(sapply(file_names, extract_number))]
reduction_methods <- c("umap", "pca")
clustering_methods <- c("gmm", "kmeans")
dfs <- readRDS(file = "sorted_news_top_year_50_comments_correct.rds")

process_and_save_comments(
  file_names, dfs, reduction_methods, clustering_methods)
```

9.4 Sentiment Analysis

```
library(reticulate)
use_virtualenv(
  "/Users/SenseiGregory/.virtualenvs/r-reticulate", required = TRUE)
vader <- import("vaderSentiment.vaderSentiment")
analyzer <- vader$SentimentIntensityAnalyzer()

get_vader_sentiment <- function(text) {
  return(analyzer$polarity_scores(text)$compound)
}
```

```r
apply_vader_to_rds <- function(file_path) {
  # Load the data frame
  df <- readRDS(file_path)

  # Check if 'comment' column exists
  if ("comment" %in% names(df)) {
    # Apply VADER sentiment analysis
    df$comment_vader_sentiment <- sapply(df$comment, get_vader_sentiment)

    # Construct the output file path
    base_name <- basename(file_path)
    output_file_path <- paste0("data/sentiment_df/", base_name)

    # Save the updated data frame
    saveRDS(df, file = output_file_path)
  } else {
    warning(paste("No 'comment' column in", file_path))
  }
}

# Apply the function to all .rds files in the specified directory
file_paths <- list.files(
  path = "data/clustering_optimized/", full.names = TRUE, pattern = "\\.rds$")
sapply(file_paths, apply_vader_to_rds)
```

9.5 Latent Dirichlet Allocation

```r
library(dplyr)
library(tidytext)
library(topicmodels)
library(tidyverse)

process_lda <- function(data) {
  data$comment_id <- seq_len(nrow(data))

  # Clean and prepare the text
  df_clean <- data %>%
    mutate(clean_body = gsub("[^[:alnum:] ]", "", comment)) %>%
    mutate(clean_body = tolower(clean_body))

  # Create a tidy text format
  tidy_text <- df_clean %>%
    unnest_tokens(word, clean_body) %>%
    select(comment_id, word)
```

```r
  # Remove stop words
  tidy_text <- tidy_text %>%
    anti_join(tidytext::stop_words, by = "word")

  # Create Document-Term Matrix
  dtm <- tidy_text %>%
    count(comment_id, word) %>%
    cast_dtm(document = comment_id, term = word, value = n)

  # Run LDA
  num_topics <- 5
  lda_model <- LDA(dtm, k = num_topics, control = list(seed = 1234))

  # Get the top terms for each topic
  topics <- tidy(lda_model, matrix = "beta")
  top_terms <- topics %>%
    group_by(topic) %>%
    top_n(20, beta) %>%
    ungroup() %>%
    arrange(topic, -beta)

  # Get the dominant topic for each document
  topics_documents <- tidy(lda_model, matrix = "gamma")
  dominant_topic <- topics_documents %>%
    group_by(document) %>%
    top_n(1, gamma) %>%
    ungroup()

  # Merge dominant topic back to original data
  dominant_topic$document <- as.integer(dominant_topic$document)
  dominant_topic <- dominant_topic %>%
    select(document, topic)

  data <- data %>%
    left_join(dominant_topic, by = c("comment_id" = "document"))

  return(list(
    all_data = data, dominant_topic = dominant_topic, top_terms = top_terms,
    dtm = dtm))
}

# Apply the function to each file
file_paths <- list.files(
  path = "data/clustering_optimized", full.names = TRUE, pattern = "\\.rds$")
results <- lapply(file_paths, function(file) {
  data <- readRDS(file)
  process_lda(data)
```

```
})
```

9.6 Analysis and Visualization

```
library(ggplot2)
library(dplyr)
library(gridExtra)
library(lubridate)
# Function to create and save a plot for each dataset
plot_sentiment_by_topic_cluster <- function(data, n, save_plot = FALSE) {

  data <- data %>% filter(!is.na(topic))

   if (is.numeric(data$date)) {
    data$date <- as.Date(as.POSIXct(
      data$date, origin = "1970-01-01", tz = "UTC"))
  } else if (is.character(data$date)) {
    data$date <- as.Date(data$date)
  }

  date_min <- min(data$date)

  if (n < length(results)) {
    next_data_date <- as.Date(as.POSIXct(min(results[[n + 1]]$all_data$date),
                                         origin = "1970-01-01", tz = "UTC"))
    date_max <- next_data_date - days(1)
  } else {
    date_max <- max(data$date)
  }

  date_min <- format(date_min, "%Y-%m-%d")
  date_max <- format(date_max, "%Y-%m-%d")

  plot_title <- paste("Sentiment Scores by Topic and Cluster -",
                      date_min, "to", date_max)
  plot <- ggplot(data, aes(x = as.factor(topic), y = comment_vader_sentiment,
                           color = as.factor(cluster))) +
    geom_point() +
    labs(title = plot_title, x = "Topic", y = "Sentiment Score",
         color = "Cluster") +
    theme_minimal() +
    theme(plot.title = element_text(size = 8))

  if (save_plot) {
    ggsave(filename = paste(
      "plots/sentiment_topic_cluster_plot_", n, ".png", sep = ""),
```

```
      plot = plot, width = 10, height = 6)
  }

  return(plot)
}

# Apply the function to each element in the results list
plots_list <- lapply(seq_along(results), function(i) {
  data <- results[[i]]$all_data
  plot_sentiment_by_topic_cluster(data, i, save_plot = TRUE)
})

plots_dir <- "plots"
if (!dir.exists(plots_dir)) {
  dir.create(plots_dir)
}

# Function to arrange and save 5 plots
arrange_save_5_plots <- function(plots, filename) {
  combined_plot <- arrangeGrob(grobs = plots, ncol = 2, nrow = 3)
  ggsave(file.path(plots_dir, filename), combined_plot, width = 10, height = 15)
}

# Function to arrange and save 4 plots
arrange_save_4_plots <- function(plots, filename) {
  combined_plot <- arrangeGrob(grobs = plots, ncol = 2, nrow = 2)
  ggsave(file.path(plots_dir, filename), combined_plot, width = 10, height = 10)
}

# Create and save the visuals
arrange_save_5_plots(plots_list[1:6], "composite_plot_1_6.png")
arrange_save_4_plots(plots_list[7:10], "composite_plot_7_10.png")
```