Natural Language Processing
Project 1.3: Oppositional thinking analysis

Erik Rubinov
erik57@campus.tu-berlin.de
Technische Universität Berlin

Niclas Tim Stoffregen
stoffregen@campus.tu-berlin.de
Technische Universität Berlin

ABSTRACT

This project addresses the critical task of differentiating between oppositional thinking and conspiratorial narratives in public health discourse through a binary text classification model. The report details the methodology for training the classification model, error analysis of its predictions, and insights into the model's limitations and potential improvements.

1 TASK 1: EXTRACT INSIGHTS FROM DATA

?? We have opted to derive the following insights from the data:

(1) Figure 1 illustrates the number of conspiracy versus critical comments within the provided dataset.
(2) Figure 2 displays the distribution of comment lengths, categorized into intervals.
(3) Figure 3 depicts the vocabulary richness in terms of unique words across both comment types.
(4) Figure 4 depicts the percentage of comments that contain at least one word identified as a slur based on a given list of "bad" words.
(5) Figure 5 demonstrates the average percentage of slur words in comments compared to normal words.
(6) Figure 6 shows the average sentiment score for the comments.
(7) Figure 7 illustrates the distribution of sentiment categorized into positive, neutral, and negative.
(8) Figure 8 shows the average number of digits contained in the comments.
(9) Figure 9 displays the average percentage of uppercase letters in the comments.

We conclude that the given data exhibits a significant imbalance, with a predominance of critical comments. Additionally, our syntactic analysis reveals that critical and conspiracy comments perform similarly, with the exceptions being the quantity of uppercase letters and comment length. Specifically, critical comments are typically 200 to 500 words long, while conspiracy comments often exceed 900 words. We can leverage both these insights later on as potential features. Our semantic sentiment analysis, however, found negligible differences between the two comment types.

2 TASK 2: PRE-PROCESSING

Pre-processing text data involves several steps aimed at cleaning and normalizing the text for easier analysis. We explain below the rationale behind selecting the following methods: lower-casing, removing specific characters and patterns, tokenization, removing stop words, and lemmatization.

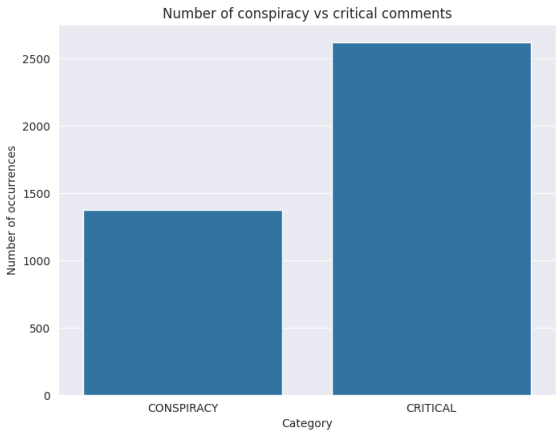

Figure 1: The number of conspiracy versus critical comments

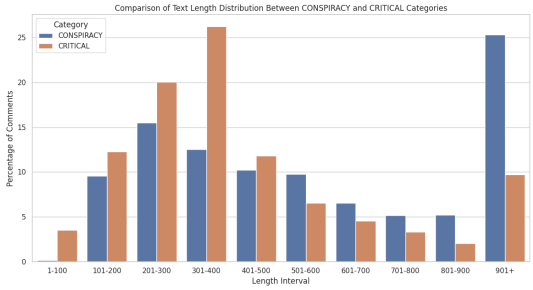

Figure 2: The distribution of the length of the comments divided into intervals.

(1) Lowercasing: This step is fundamental in reducing the dimensionality of text data and avoiding redundancy.
(2) Removing Numerals and Specific Punctuation: Numerals may not carry meaningful information in many text analysis tasks, and specific punctuation like colons, hyphens, and apostrophes can be removed to simplify the text without losing significant information.
(3) Removing URLs: URLs are typically removed because they are not usually relevant to the textual content's semantic meaning.
(4) Removing Extra Whitespace: Normalizing whitespace ensures that tokens are separated by single spaces, which facilitates easier tokenization.

Technische Universität Berlin, 2024
2024.

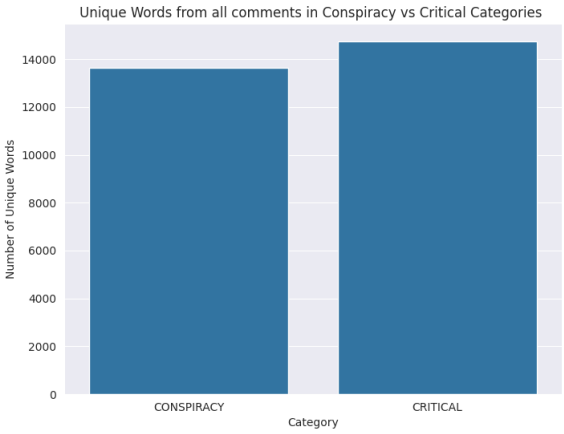

Figure 3: The number of unique words in the comments.

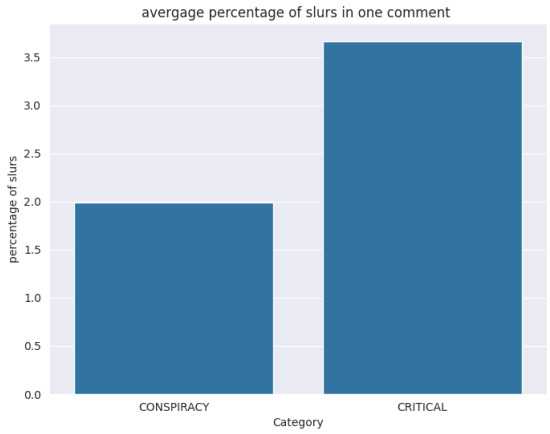

Figure 5: The average occurrence of slurs in each comment throughout the dataset

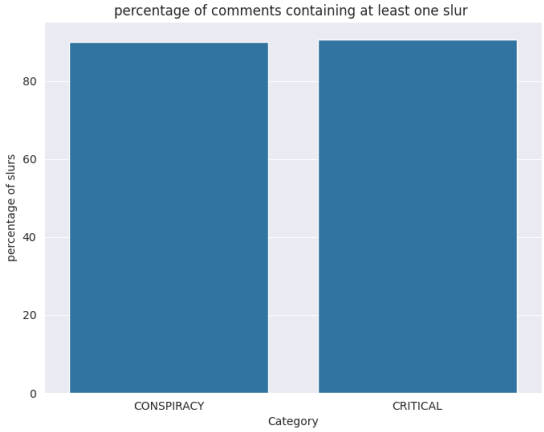

Figure 4: The percentage of comments that contain at least one slur.

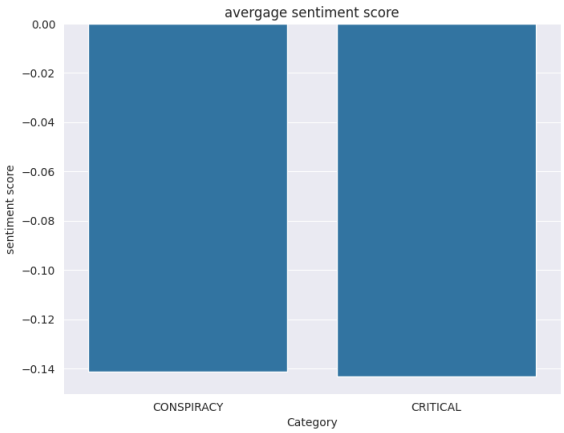

Figure 6: The average sentiment score of the comments.

(5) Removing Special Characters: Special characters often do not contribute to the semantic meaning of the text and can be removed to clean the data.
(6) Removing Floating Point Numbers: Similar to the removal of general numerals, floating point numbers are also removed to ensure that numerical data does not interfere with text analysis. Prior to the removal of numerical data, we conducted an analysis to determine if specific numbers were predominantly associated with one category. However, no discernible patterns were identified.
(7) Removing Specific Words: Removing specific words like "com" (often part of URLs or email addresses) helps in cleaning the text further by removing irrelevant content
(8) Tokenization: Tokenization splits the text into individual words or tokens. This step is crucial for many NLP tasks, as it converts a text string into a list of meaningful components.
(9) Removing Stop Words: Stop words (common words like 'the', 'is', 'in') are removed because they typically do not carry significant meaning and can clutter the analysis.
(10) Lemmatization: Lemmatization reduces words to their base or dictionary form (e.g., 'running' to 'run'). This helps in normalizing the text and reducing the complexity by treating different forms of a word as a single term.

3 TASK 3: TEXT CLASSIFICATION

For the third task, we constructed various pipelines to assess the interplay between different components. The components are as follows:

(1) Two different pre-processing function
(2) Two different vectorizer
(3) Two different models
(4) Two new features

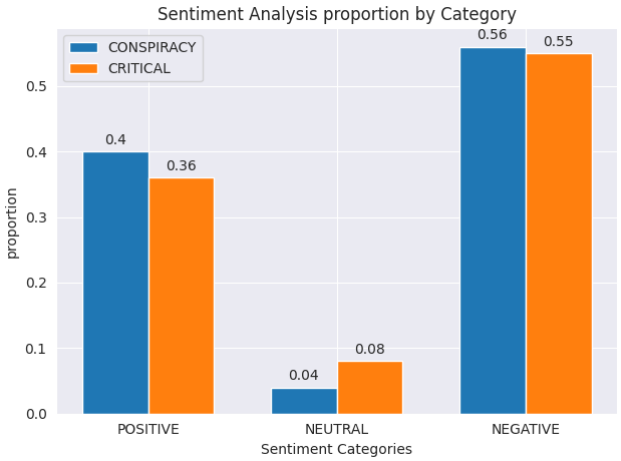

Figure 7: The sentiment classified into positive, neutral and negative.

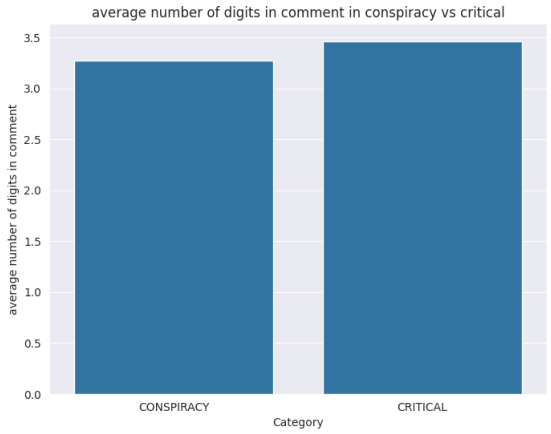

Figure 8: The average number of digits in the comments.

(5) Three different n-gram ranges

3.1 Pre-Processing

We chose to employ two pre-processing function. The simpler one is a streamlined version that focuses on the essential preprocessing steps: Tokenization, Lemmatization, Stemming.
The advanced function is more comprehensive and includes several all the steps that we mention in section 2. The advanced function is more suitable for scenarios where data cleanliness is crucial and where the computational cost is less of a concern compared to the quality of input data. The simpler one might be favored in scenarios where quick, less computationally expensive pre-processing is required, and the text data is already relatively clean or when very high precision in text normalization is not necessary. Both preprocessing functions, lemmatization and stemming, can be disabled, allowing for additional experimental flexibility

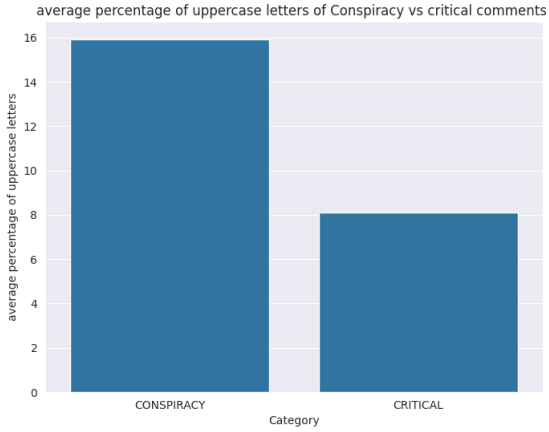

Figure 9: The average percentage of uppercase letters.

3.2 Vectorizer

We employ a TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer and a CountVectorizer.
The CountVectorizer transforms text into a matrix of token counts, where each row represents a document and each column represents a term from the corpus. The values in the matrix indicate the frequency of terms in each document.
The TF-IDF vectorizer builds upon the CountVectorizer by weighting the term frequencies by their inverse document frequencies. This adjustment downplays common terms that appear across many documents and highlights rarer, more significant terms, making it particularly useful for identifying the most relevant terms in each document.

3.3 Models

We employed a naive Bayes model and a feed-forward neural network. We used the `sklearn`'s `MultinomialNB` and `MLPClassifier`. For the former, we used the default parameters. For the latter, we employed a configuration with the number of neurons in the i-th hidden layer reduced by half.

3.4 Addition Features: Length and Upper-casing

As described in section 2, we found that comment length and the amount of upper-cased letters differ between the categories.

3.4.1 Comment Length. The length of a comment is a straightforward quantitative measure that can provide valuable insights independently of the word content itself. Conspiracy theories often involve intricate and elaborate explanations that attempt to tie together various unrelated events, facts, or perceptions into a coherent narrative that challenges the official or mainstream account of events. In Task 1, we demonstrated that comments categorized as conspiratorial were, on average, 75% longer than those labeled as critical. CountVectorizer counts the number of times each word appears in the document but doesn't retain information about the total number of words in the document unless you specifically look at the sum of all counts, which still doesn't tell you about

non-counted words (like those filtered out by stop words). TfidfVectorizer weights the word counts by their inverse document frequency, adjusting for words that are too common across documents. This method focuses more on the significance of words rather than their quantities and thus does not inherently reflect the length of the comments. Therefore we save the length of the comment to not loose this information. By categorizing the length of the comment into three distinct categories, we reduce the variability and noise that might come from minor fluctuations in the percentage values.

3.4.2 Upper-casing. Texts with a high percentage of uppercase letters may convey stronger emotions or intensity. For instance, users might use uppercase text to express anger, excitement, or urgency. Classifying texts based on the usage of uppercase letters could help in understanding the emotional context or sentiment of the text. In Task 1, we also found that the average percentage of uppercase letters in conspiracy comments was twice as high compared to critical comments. During the vectorization the information of the percentage of uppercase letters is letters since all letters get lower-cased. Therefore we added an extra parameter that saves the upper-case percentage of each comment into three different levels. For the same reasons as for the comment length parameter we use discretization by categorizing the comment length in three categories.

3.5 N-gram Ranges

When used with vectorizers such as in our TF-IDF, n-grams help create a more nuanced and detailed representation of the text data. For example, distinguishing between "New York" (as a place) and "new york" (as separate words in a different context). In our work we experimented with unigrams, bigrams and a mixture of both. We hoped to get a richer feature set compared to just using unigrams.

3.6 Evaluation

All of our experiments have been conducted on a MacBook Pro with Apple's M2 Max and 32GB of RAM. Our main metric to evaluate is the F1-Scored issued in the classification task.

3.6.1 Pre-Processing. Figure 10 depicts the performance depending on the employed pre-processing function. As both perform equally well, we can resort to the basic function in future work, since it saves on time and is less computationally intensive.

3.6.2 Lemmatization and Stemming . Figure 11 illustrates the differences in F1 scores when Lemmatization and/or Stemming are applied. These preprocessing configurations were evaluated in conjunction with a CountVectorizer using a unigram model and the basic preprocessing function. Stemming appears to decrease the F1 score by 1%. Similar outcomes were observed when employing the TfidfVectorizer.

3.6.3 Vectorizers Models. We observe in figure 12 hardly any difference in performance when switching the vectorizers and models.

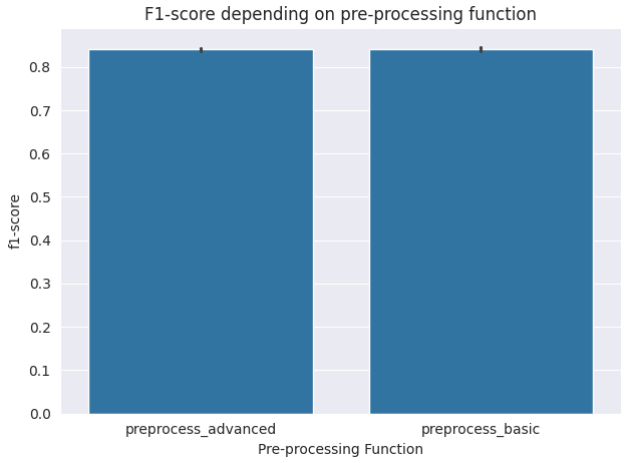

Figure 10: F1-Score depending on the pre-processing function.

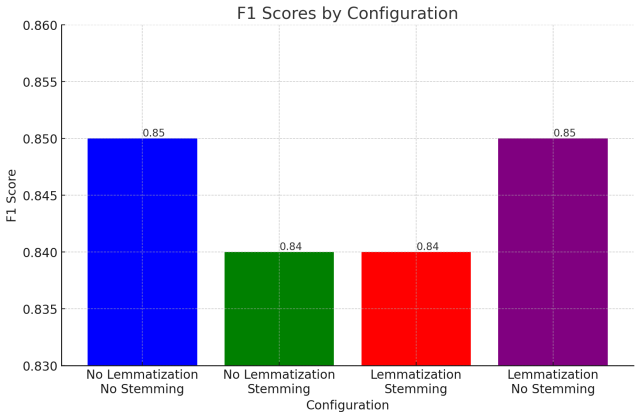

Figure 11: F1-Score depending Lemmatization and Stemming.

3.6.4 Addition Features: Length and Upper-casing. As depicted in figure 13, we observe that pipelines incorporating the added features perform slightly better than those with no additional features.

3.6.5 N-gram Ranges. Contrary to our expectations, we found that sticking to the default unigrams (1,1) returns the most promising results as can be seen in figure 14. For this reason and to save on the extensive computational time, we removed differing n-gram ranges from our notebook.

All in all, the best best performance was produced by a MLPlassifier, with a CountVectorizer, the basic pre-processing function the the comment length as an additional feature. It resulted in a f1-score of 0.86. But overall, it can be said that the f1-score does differ only by 0.04 throughout all pipeline runs.

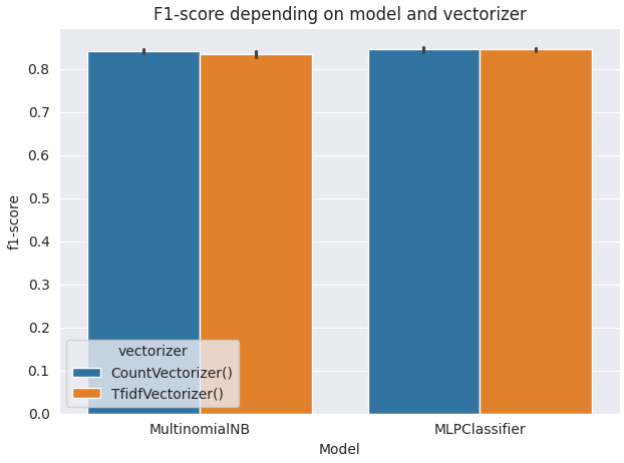

Figure 12: F1-Score depending on the vectorizer and model that was chosen.

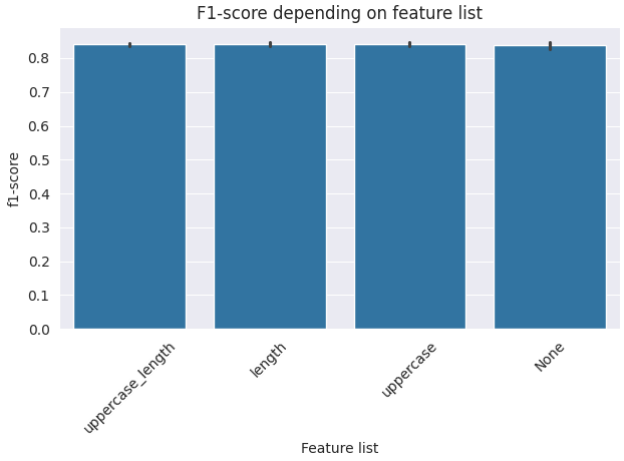

Figure 13: F1-Score depending on the feature list. "uppercase_length" takes into account both the length and uppercase feature.

3.7 Error Analysis

In this section, we dive deeper into some of the errors the models made. Both models classfy roughly 120 comment instances wrongly. We conducted a manual analysis of the matter.

3.7.1 Naive Bayes Model. The best performing Naive Bayes model was utilized with TfidfVectorizer, the advanced preprocessing function that excludes lemmatization and stemming, and includes comment length and uppercase count as additional features.

3.7.2 Feed-Forward Neural Network. We find that the best performing FNN was used with a CountVectorizer, the simpler preprocessing function, neither lemmatization nor stemming and comment length as the only feature passed.

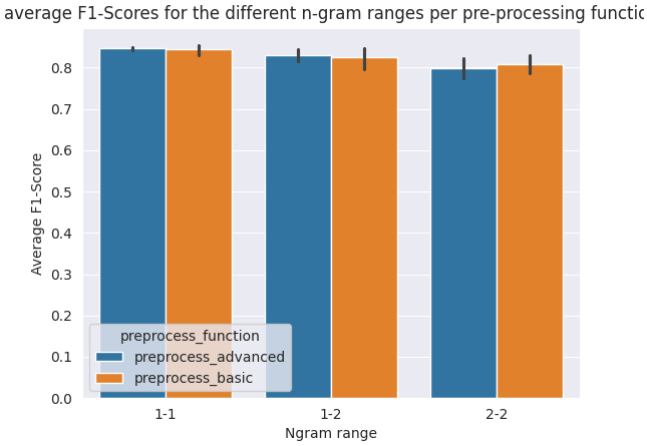

Figure 14: The performance of the models worses with larger n-gram ranges.

Our manual analysis found that both models tend to misclassify ambiguous instances such as "*BQQQQQQQMFauCGI opens up the possibility that the COVID - 19 vaccine could be making people more likely to be infected by the virus . “ This would not be the first time , if it happened , that a vaccine that looked good in initial safety actually made people worse . " - THOUGHTS ? SHARE ! ! https :// t . me / QNewsOfficialTV*" which was misclassified as a conspiracy comment but the given data labels it as critical. Another example is "*BQQQQQQQMMMDoctor provides evidence using vials of how sexual intercourse between an unvaccinated person and a vaccinated person can make an unvaccinated persons blood contaminated - THOUGHTS ? SHARE ! ! https :// t . me / QNewsOfficialTV*" that was misclassified as critical but seemts to be a conspiracy comment.

4 TASK 4: TEXTUAL SIMILARITY

For the vector representation, we employed the same GloVe pretrained embeddings and utilized the `preprocess_advanced` preprocessing function, which was also applied in Task 3. The heatmap with the corresponding cosine similarities between 15 comments can be found in figure 15

Model	Epoch	Accuracy
BERT	1	0.9418
BERT	2	0.9561
BERT	3	0.9571
RoBERTa	1	0.4948
RoBERTa	2	0.6397
RoBERTa	3	0.6473

Table 1: Accuracy values for BERT and RoBERTa models across different epochs

Table 2: BERT Binary Classification Report

	Precision	Recall	F1-score	Support
CONSPIRACY	0.94	0.97	0.96	519
CRITICAL	0.97	0.94	0.96	530
Accuracy			0.96	1049
Macro avg	0.96	0.96	0.96	1049
Weighted avg	0.96	0.96	0.96	1049

Table 3: RoBERTa Binary Classification Report

	Precision	Recall	F1-score	Support
CONSPIRACY	0.62	0.72	0.67	519
CRITICAL	0.68	0.58	0.62	530
Accuracy			0.65	1049
Macro avg	0.65	0.65	0.65	1049
Weighted avg	0.65	0.65	0.65	1049

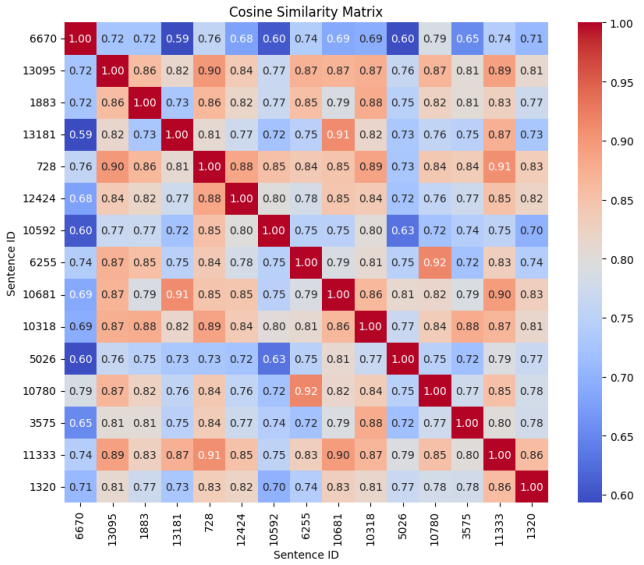

Figure 15: Cosine Similarity Matrix between 15 random comments.

5 TASK 5: BONUS TASK - TEXTUAL SIMILARITY

We employed two pre-trained models, BERT and RoBERTa, to perform the classification task. Our approach involved the following steps:

(1) **Data Preparation**
- Loaded and preprocessed the dataset, which included encoding the text labels.
- Split the data into training and testing sets.

(2) **Model Selection**
- Fine-tuned the models for binary classification, ensuring careful selection of hyperparameters to avoid overfitting.

(3) **Training Process**
- Used the Hugging Face `transformers` library to fine-tune the models.
- Trained the models for three epochs with a learning rate of 5e-5.
- Evaluated the models on the test dataset.

(4) **Performance Evaluation**
- Evaluated the models using precision, recall, F1-score, and accuracy metrics.
- Compared the performance of the two models.

The BERT model significantly outperformed the RoBERTa model. We see in table **??** and **??** that BERT achieved an F1-score of 0.96 for both classes, resulting in an overall accuracy of 96%. On the other hand, RoBERTa achieved an F1-score of 0.67 for "CONSPIRACY" and 0.62 for "CRITICAL," with an overall accuracy of 65%. The results indicate that BERT is more suitable for this specific binary classification task on the our dataset.