Oppositional Thinking Analysis

This is Short Title of the paper, used in page headers

This is the subtitle of the paper, this document both explains and embodies the submission format for authors using Word

Dimitrij Schulz

TU Berlin, dmtschulz@gmail.com

This report addresses oppositional thinking analysis through text classification, focusing on distinguishing critical from conspiratorial narratives in a dataset of 4000 email instances. Tasks include data insights extraction, pre-processing steps selection, text classification using Naïve Bayes and Neural Network models, as well as semantic textual similarity computation. Results underscore the effectiveness of different approaches in classifying and analyzing discourse, contributing insights into misinformation detection.

CCS CONCEPTS • Computing methodologies~Artificial intelligence~Natural language processing~Information extraction

Additional Keywords and Phrases: text classification, vectorization methods, Naïve Bayes, feedforward neural network, preprocessing pipelines, TF-IDF, Count Vectorizer, dataset imbalance, textual similarity, distributional semantics, cosine similarity, NLP, conspiracy narratives, critical thinking narratives, word embeddings

ACM Reference Format:

First Author's Name, Initials, and Last Name, Second Author's Name, Initials, and Last Name, and Third Author's Name, Initials, and Last Name. 2018. The Title of the Paper: ACM Conference Proceedings Manuscript Submission Template: This is the subtitle of the paper, this document both explains and embodies the submission format for authors using Word. In Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY. ACM, New York, NY, USA, 10 pages. NOTE: This block will be automatically generated when manuscripts are processed after acceptance.

1 EXTRACTING INSIGHTS FROM DATA

Understanding the distribution and characteristics of text data is foundational in any NLP endeavor. In this initial task, I aim to extract meaningful insights from the dataset through statistical analysis and visual representations. Key objectives include examining variations in text length across different categories, identifying the number of unique words per category, and exploring other pertinent metrics that shed light on the nature of oppositional thinking within the corpus.

In this task, I begin by loading and extracting data from the 'Oppositional_thinking_analysis_dataset.json' file, which contains 4000 instances categorized into spam and normal email texts. Using Python and Pandas, I convert the extracted data into a DataFrame format for structured analysis. The initial step involves exploring fundamental characteristics of the dataset, such as text length distributions and the average number of unique words per category. Through visualizations like histograms and bar charts, I aim to uncover key insights that differentiate oppositional thinking narratives from other types of discourse. This foundational analysis not only sets the stage for subsequent tasks but also provides essential context for understanding the dataset's composition and variability.

My analysis of the dataset reveals several critical distinctions between conspiracy and critical thinking narratives. Conspiracy texts exhibit richer lexical diversity, averaging 80 unique words compared to 60 in critical texts. Notably, "5G" or "5 G" terms are significantly more prevalent in conspiracy texts, appearing in 6% of instances compared to nearly 0% in critical texts. Conversely, mentions of "Bill Gates" are markedly higher in conspiracy texts at 4%, contrasting sharply with a mere 1% in critical texts. Moreover, conspiracy texts tend to employ more uppercase words and exclamation marks on average, indicative of heightened emphasis or emotion. Another striking difference is the prevalence of sentences around 100 characters in length, with conspiracy texts containing approximately 1000 such instances compared to 300 in critical texts. These findings underscore distinct linguistic patterns and thematic focuses that differentiate oppositional thinking narratives within the dataset.

2 PREPROCESSING

Effective pre-processing of textual data is essential for optimizing the dataset of 4000 email instances categorized into conspiracy and critical thinking narratives. In this task, I focus on key pre-processing steps like tokenization, stopwords removal, lemmatization, and others. These techniques are selected to standardize text representations and enhance the clarity of distinctions between narrative categories. My choices are informed by their ability to reduce noise and improve data quality, aligning with the specific goals of the oppositional thinking analysis. I acknowledge the importance of these steps in preparing the dataset for subsequent modeling tasks while considering their relevance over other available pre-processing techniques.

For this project, I have implemented a series of pre-processing steps tailored to enhance the quality and relevance of the dataset of 4000 email instances categorized into conspiracy and critical thinking narratives. My approach includes essential techniques such as lowercase conversion, removal of URLs, punctuation normalization (excluding exclamation marks), and tokenization using NLTK's word_tokenize function. I opted for lemmatization using WordNetLemmatizer to reduce words to their base forms, which aids in standardizing word representations across texts. Additionally, I selectively removed stopwords from the English language using NLTK's stopwords corpus to eliminate common but non-informative words. These chosen steps are aimed at improving the dataset's coherence and facilitating clearer distinctions between narrative categories. Conversely, I decided against stemming techniques, which can overly simplify word forms and potentially reduce interpretability in this specific context.

3 TEXT CLASSIFICATION

In this task, I embark on a comprehensive evaluation of models designed for binary classification of narratives within the dataset. The primary objectives include preparing the data by splitting it into training and test sets, ensuring balanced representation through undersampling techniques. I initiate the process by training a Naïve Bayes model on the training data and subsequently evaluating its performance on the designated test set. A critical aspect of this task involves exploring the impact of various vectorization models, such as Count Vectorizer and TF-IDF, on enhancing the Naïve Bayes model's predictive accuracy. Additionally, I scrutinize different pre-processing pipelines, including strategies with and without stopwords and with and without lemmatization, to gauge their influence on model performance. Error analysis will be conducted to pinpoint areas where the Naïve Bayes model may falter, elucidating potential shortcomings in distinguishing between critical discourse and conspiratorial narratives. Furthermore, I augment my investigation by training a feed-forward neural network model and comparing its performance, particularly focusing on metrics like the F1 score. This comparative analysis between the Naïve Bayes and neural network models aims to derive insights into their respective strengths and limitations, culminating in a nuanced evaluation of their efficacy for this binary classification task.

3.1 Dataset Preparation

The original dataset was initially divided into an 80% train set comprising 3200 instances and a 20% test set comprising 800 instances. In the original train set, class counts were unevenly distributed with 1103 instances labeled as class 1 (conspiracy messages) and 2097 instances as class 0 (critical messages). Similarly, the test set had 524 instances of class 0 and 276 instances of class 1. To address this class imbalance, undersampling was applied, resulting in a reduced train set size of 2450 instances and an expanded test set size of 1550 instances. Post-undersampling, the train set now consists of 1103 instances of class 1 and 1347 instances of class 0, while the test set comprises 1274 instances of class 0 and 276 instances of class 1. These adjustments were crucial for ensuring a more balanced representation of classes in both training and testing phases, enhancing the reliability and generalizability of subsequent model training and evaluation processes.

3.2 Naïve Bayes Model Training

For the Naïve Bayes training, I initialized dictionaries for results and misclassifications and created a directory to save results. Data was loaded and split into train and test sets, with class distributions recorded. To address class imbalance, I applied undersampling, balancing the datasets by moving samples of the 'CRITICAL' category (labeled as 0).

Each data file and vectorizer combination was used to train and evaluate the Naïve Bayes model. This process enabled a comprehensive comparison of pre-processing and vectorization strategies, highlighting their impact on model performance. Table 1 summarizes the results of various vectorization models and pre-processing pipelines on the performance of the Naïve Bayes model.

Table 1: Performance Comparison of Naïve Bayes Model Using Different Pre-processing Pipelines and Vectorization Methods

Data File	Vectorizer	F1 Score (weighted)	Misclassified Test Examples
With stopwords with lemmatization	TfidfVectorizer1	0.8942	165
With stopwords with lemmatization	TfidfVectorizer2	0.8688	208
With stopwords with lemmatization	TfidfVectorizer3	0.8478	229
With stopwords with lemmatization	CountVectorizer1	0.8307	288
With stopwords with lemmatization	CountVectorizer2	0.8255	298
With stopwords with lemmatization	CountVectorizer3	0.8082	324
Without stopwords with lemmatization	TfidfVectorizer1	0.8963	160
Without stopwords with lemmatization	TfidfVectorizer2	0.8616	208
Without stopwords with lemmatization	TfidfVectorizer3	0.8134	262
Without stopwords with lemmatization	CountVectorizer1	0.8612	230
Without stopwords with lemmatization	CountVectorizer2	0.8432	257
Without stopwords with lemmatization	CountVectorizer3	0.7941	324
With stopwords without lemmatization	TfidfVectorizer1	0.8869	178
With stopwords without lemmatization	TfidfVectorizer2	0.8677	212
With stopwords without lemmatization	TfidfVectorizer3	0.8482	230
With stopwords without lemmatization	CountVectorizer1	0.8338	283
With stopwords without lemmatization	CountVectorizer2	0.8179	312
With stopwords without lemmatization	CountVectorizer3	0.7982	344
Without stopwords without lemmatization	TfidfVectorizer1	0.8956	162
Without stopwords without lemmatization	TfidfVectorizer2	0.8576	217
Without stopwords without lemmatization	TfidfVectorizer3	0.8186	261
Without stopwords without lemmatization	CountVectorizer1	0.8584	236
Without stopwords without lemmatization	CountVectorizer2	0.8379	266
Without stopwords without lemmatization	CountVectorizer2	0.8002	316

TfidfVectorizer1 is a unigram model, TfidfVectorizer2 includes bigrams, and TfidfVectorizer3 includes trigrams. Similarly, CountVectorizer1 is a unigram model, CountVectorizer2 includes bigrams, and CountVectorizer3 includes trigrams.

3.2.1Comparison of the Impact of different vectorization Models

The comparison of different vectorization models revealed that TF-IDF vectorizers consistently outperformed Count Vectorizers in terms of the F1 score. Among the TF-IDF vectorizers, TfidfVectorizer1, which uses unigrams, achieved the highest F1 score of 0.8963 with the least number of misclassified examples. In contrast, TfidfVectorizer3, using a combination of n-grams, showed a lower F1 score and higher misclassification rate. For Count Vectorizers, CountVectorizer1, which uses unigrams, performed better than other configurations but still lagged behind the TF-IDF vectorizers.

3.2.2Compare the Impact of different pre-processing Pipelines

The impact of different pre-processing pipelines was significant on the model’s performance. Models trained on data preprocessed without stop words and with lemmatization generally performed better, as seen with the highest F1 score of 0.8963 achieved using TfidfVectorizer1. Conversely, the pipeline including stop words and without lemmatization resulted in lower performance, indicating that removing stop words and applying lemmatization can enhance the model's ability to accurately classify messages.

3.2.3Naïve Bayes Error Analysis

The best performance was achieved using the dataset preprocessed without stopwords and with lemmatization in conjunction with TfidfVectorizer1 (unigram). This configuration resulted in the highest F1 score of 0.8963 and the fewest misclassified examples, with 160 out of 1550. This suggests that removing stopwords and applying lemmatization effectively reduces noise and enhances the model's ability to generalize better, especially when using a TF-IDF representation of unigrams.

The worst performance was observed with the dataset preprocessed with stopwords and without lemmatization paired with CountVectorizer3 (trigram). This combination yielded the lowest F1 score of 0.7982 and the highest number of misclassified examples, with 344 out of 1550. The inclusion of stopwords and lack of lemmatization likely introduced significant noise and redundancy, exacerbating the challenge of accurately capturing the semantic content of the messages.

3.3 Feedforward Neural Network Model Training

For training a Feedforward Neural Network (FFNN) model, I defined several key hyperparameters: two hidden layers with sizes 64 and 32, an output size of 1, a dropout rate of 0.4, a learning rate of 0.001, a batch size of 32, and 10 epochs for training. The model was configured to run on a GPU if available, otherwise on a CPU. Each data file was preprocessed using different vectorizers, and the resulting datasets were split into training and testing sets, which were then loaded into PyTorch DataLoader objects to facilitate batch processing during training. The FFNN model architecture consisted of two fully connected layers with ReLU activations and dropout layers for regularization, followed by a final fully connected layer with a sigmoid activation function to output the prediction. The model was trained and evaluated using the binary cross-entropy loss function and the Adam optimizer on a GTX 1060 with a 6 GB RAM graphics card. Table 2 summarizes the results of various vectorization models and pre-processing pipelines on the performance of the FFNN model.

Table 2: Performance Comparison of FFNN Model Using Different Pre-processing Pipelines and Vectorization Methods

Data File	Vectorizer	F1 Score (weighted)	Misclassified Test Examples
With stopwords with lemmatization	TfidfVectorizer1	0.8424	268
With stopwords with lemmatization	TfidfVectorizer2	0.8424	267
With stopwords with lemmatization	TfidfVectorizer3	0.8159	310
With stopwords with lemmatization	CountVectorizer1	0.8652	223
With stopwords with lemmatization	CountVectorizer2	0.8707	211
With stopwords with lemmatization	CountVectorizer3	0.8556	228
Without stopwords with lemmatization	TfidfVectorizer1	0.8471	257
Without stopwords with lemmatization	TfidfVectorizer2	0.8455	251
Without stopwords with lemmatization	TfidfVectorizer3	0.7910	324
Without stopwords with lemmatization	CountVectorizer1	0.8698	213
Without stopwords with lemmatization	CountVectorizer2	0.8527	237
Without stopwords with lemmatization	CountVectorizer3	0.8112	264
With stopwords without lemmatization	TfidfVectorizer1	0.8497	254
With stopwords without lemmatization	TfidfVectorizer2	0.8226	305
With stopwords without lemmatization	TfidfVectorizer3	0.8122	317
With stopwords without lemmatization	CountVectorizer1	0.8492	254
With stopwords without lemmatization	CountVectorizer2	0.8669	216
With stopwords without lemmatization	CountVectorizer3	0.8429	232
Without stopwords without lemmatization	TfidfVectorizer1	0.8597	234
Without stopwords without lemmatization	TfidfVectorizer2	0.8366	267
Without stopwords without lemmatization	TfidfVectorizer3	0.8009	314
Without stopwords without lemmatization	CountVectorizer1	0.8489	254
Without stopwords without lemmatization	CountVectorizer2	0.8554	231
Without stopwords without lemmatization	CountVectorizer3	0.8121	290

TfidfVectorizer1 is a unigram model, TfidfVectorizer2 includes bigrams, and TfidfVectorizer3 includes trigrams. Similarly, CountVectorizer1 is a unigram model, CountVectorizer2 includes bigrams, and CountVectorizer3 includes trigrams.

3.3.1Comparison of the Impact of different vectorization Models

Across the datasets analyzed, the choice of vectorization method—TfidfVectorizer versus CountVectorizer—shows distinct impacts on classification performance. When using TfidfVectorizer, which weighs terms based on their frequency-inverse document frequency, the misclassification rates tend to vary. For instance, in datasets with stopwords and lemmatization, TfidfVectorizer consistently resulted in higher misclassifications compared to CountVectorizer, particularly noticeable with n-gram sizes of 2 and 3. In contrast, CountVectorizer, which counts term frequencies directly, generally exhibited lower misclassification rates across all datasets, showcasing its robustness in capturing meaningful distinctions in text data. This comparison underscores the importance of considering the inherent characteristics of each vectorization model when optimizing text classification tasks.

3.3.2Compare the Impact of different pre-processing Pipelines

Comparing different preprocessing pipelines reveals significant impacts on text classification results. Datasets with stopwords and lemmatization generally exhibit higher misclassification rates with TfidfVectorizer compared to CountVectorizer. The choice of n-gram range within each vectorization model also influences performance, with larger n-grams sometimes increasing misclassification errors.

3.4 Comparing Naïve Bayes and feed-forward Model Performance

The Naïve Bayes model consistently outperforms the Feed-Forward Neural Network across all vectorization strategies and preprocessing pipelines in terms of F1 score and misclassification rates. Specifically, Naïve Bayes achieves higher F1 scores with generally fewer misclassified examples compared to FFNN. For instance, with stopwords and lemmatization, Naïve Bayes using TfidfVectorizer1 achieves an F1 score of 0.8942 with 165 misclassified examples, whereas FFNN achieves an F1 score of 0.8424 with 268 misclassified examples. Similarly, across other configurations, Naïve Bayes consistently demonstrates superior performance metrics.

One possible reason for the FFNN's comparatively poorer performance could be attributed to the size of the training dataset. FFNNs typically require larger amounts of data to effectively learn complex patterns and generalize well to unseen examples. In this context, the dataset size may not have been sufficient to fully capitalize on FFNN's potential, leading to suboptimal performance when compared to Naïve Bayes. These results underline the importance of dataset size and model suitability in achieving robust performance in text classification tasks. Therefore, when deciding between Naïve Bayes and FFNN for similar tasks, practitioners should consider both model performance and the adequacy of the training dataset.

4 TEXTUAL SIMILARITY

In this task, the objective is to assess textual similarity using distributional semantics at the sentence level. Specifically, 15 random instances will be selected, and the average of word vectors will be employed as a measure of semantic similarity between messages. The cosine similarity metric will be computed to quantify the degree of similarity between pairs of randomly chosen sentences. This approach provides insights into how effectively distributional semantics can capture semantic relationships between texts based on their word embeddings. The findings from this analysis will be detailed in the report, highlighting the computed cosine similarities and their implications for understanding textual coherence and similarity. I randomly chose the following 15 sentences:

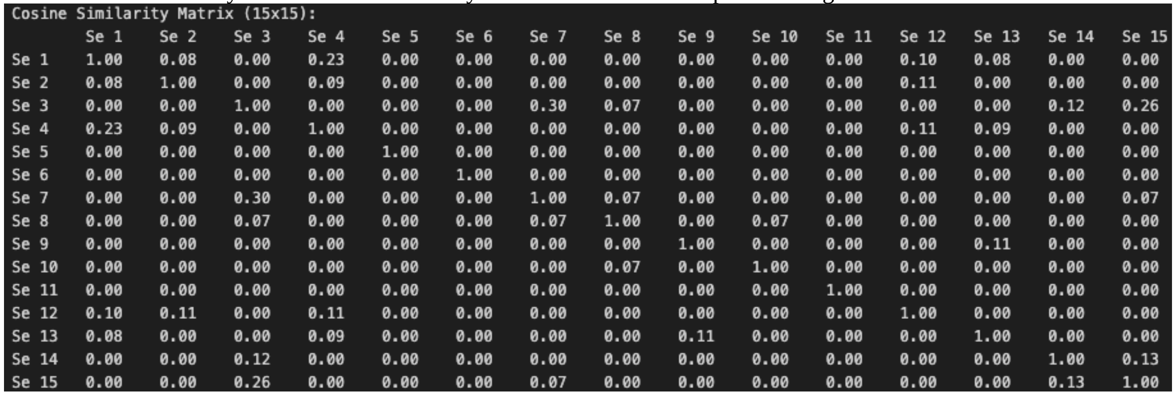
1. coronavirus death joe biden first ten month office surpass death toll trump
2. nolte bidensanity unvaccinated illegal alien shipped dozen state coronavirus fear rage
3. new world order want people dead rest slave make sure dont child satanist want eat
4. exclusive govkristi noem joe biden rationing life saving coronavirus medicine political reason
5. short news clip alleged swine flu pandemic exaggerated position power maximise profit
6. postal worker union role federal government mandate vaccination
7. new world order ally uniparty want destroy united stateswe must let themstay alert keep head swivel
8. quote yetthere official definition long covid diagnostic criterion condition new zealand ministry health
9. report climate change pose threat much planetincluding american sovereignty
10. ukraine drama staged event like covid wasreal limited scale war blown proportion pub four
11. stanley plotkin admits use aborted baby vaccinesyou see contempt oozing jew ca nt youfull deposition
12. carlson america lost religion replaced cult coronavirus
13. greta thunberg biden admin ready act seriously need climate crisis
14. made take pig vaccine would think people would upset satanist resistance stop satan info saveusnoworguk
15. vaccine bio chemical weapon warn people got electron microscope image antenna contaminant want dead

Labels of the sentences in the corresponding order are: 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, where 0 means critical and 1 means conspiracy.

First, I iterated through each sentence to tokenize it into individual words and add each unique word to a set to ensure that no duplicates are included. For each sentence, I initialized a frequency vector of length equal to the number of unique words, initially filled with zeros. I then tokenized the sentence into individual words and for each word, found its index in the list of unique words, incrementing the corresponding position in the frequency vector. This process resulted in a vector that represents the count of each unique word in the sentence. Next, for each pair of sentences, I retrieved their corresponding frequency vectors and computed the dot product of the two vectors. I also computed the norm (magnitude) of each vector. Using these values, we calculate the cosine similarity between 2 sentences A and B with the formula:

$$\frac{\text{cosine similarity}(\mathrm{A}, \mathrm{B}) = (A \bullet B)}{\| \mathrm{A} \| \| \mathrm{B} \|}$$

The cosine similarity between these randomly selected sentences in depicted in Figure 1.



```
Cosine Similarity Matrix (15x15):
        Se 1    Se 2    Se 3    Se 4    Se 5    Se 6    Se 7    Se 8    Se 9    Se 10   Se 11   Se 12   Se 13   Se 14   Se 15
Se 1    1.00    0.08    0.00    0.23    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.10    0.08    0.00    0.00
Se 2    0.08    1.00    0.00    0.09    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.11    0.00    0.00    0.00
Se 3    0.00    0.00    1.00    0.00    0.00    0.00    0.30    0.07    0.00    0.00    0.00    0.00    0.00    0.12    0.26
Se 4    0.23    0.09    0.00    1.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.11    0.09    0.00    0.00
Se 5    0.00    0.00    0.00    0.00    1.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
Se 6    0.00    0.00    0.00    0.00    0.00    1.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
Se 7    0.00    0.00    0.30    0.00    0.00    0.00    1.00    0.07    0.00    0.00    0.00    0.00    0.00    0.00    0.07
Se 8    0.00    0.00    0.07    0.00    0.00    0.00    0.07    1.00    0.00    0.07    0.00    0.00    0.00    0.00    0.00
Se 9    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    1.00    0.00    0.00    0.00    0.11    0.00    0.00
Se 10   0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.07    0.00    1.00    0.00    0.00    0.00    0.00    0.00
Se 11   0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    1.00    0.00    0.00    0.00    0.00
Se 12   0.10    0.11    0.00    0.11    0.00    0.00    0.00    0.00    0.00    0.00    0.00    1.00    0.00    0.00    0.00
Se 13   0.08    0.00    0.00    0.09    0.00    0.00    0.00    0.00    0.11    0.00    0.00    0.00    1.00    0.00    0.00
Se 14   0.00    0.00    0.12    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    1.00    0.13
Se 15   0.00    0.00    0.26    0.00    0.00    0.00    0.07    0.00    0.00    0.00    0.00    0.00    0.00    0.13    1.00
```

Figure 1: Cosine Similarity Matrix for 15 Sentences, Se1 (Sentence 1) till Se15 (Sentence 15).