Analyzing Public Opinion Bias on Pandemic through Oppositional Thinking Analysis

Siyu Xiao
Technische Universität Berlin
Berlin, Deutschland
xiao.siyu@tu-berlin.de

1 Introduction

Oppositional thinking analysis can be formulated as a text classification task (like spam filtering), where conspiracy theories and critical thinking narratives represented as a category. This distinction is vital because labeling a message as conspiratorial when it is only oppositional could drive those who were simply asking questions into the arms of the conspiracy communities.

This project is a binary classification task differentiating between:

(1) `critical` messages that question major decisions in the public health domain, but do not promote a conspiracist mentality;
(2) messages that view the pandemic or public health decisions because of a malevolent `conspiracy` by secret, influential groups.

2 Dataset

Each instance in the Json dataset corresponds to a dictionary that contains tokenized text, the binary category, and the span annotations.

`Oppositional_thinking_analysis_dataset.json`

For this project, the span annotations are ignored in the training procedure, but are observed in the initial analysis. The text and the category are the input and output the model. The data includes 4000 instances of spam and normal (ham) email texts.

3 Implementation

All class and functions are executed in `Jupyter Notebook`, using `pandas` as fundamental data frame for data processing, which is suitable for LM and dealing with multiple features in the dataset.

For visualization, beside basic `matplotlib.pyplot` and `seaborn`, `wordcloud` and `plotly` are used for specialized representation for text corpus and 3D conprehension of data structure.

3.1 Extract insights from data

Initial analysis before pre-processing is vital for further procedure. The original data is read from `.json` file and stored in `pandas DataFrame` Fig.1. Looking at the features/columns, each unique text has one `id` and category: `Conspiracy` and `Critical`, representing the 2 sides of oppsitional thinking. Annotations column is not the focus but shows some insights. `Spacy tokens` are tokenized texts, as the input of the next pre-processing task.

To have a general view of the categories, the distribution over each category is captured and plotted below Fig.2 through two different numeric methods: the length of each text, and the number of unique words.

id	text	category	annotations	spacy_tokens	text_length	unique_words
5206	THIS IS MASS...	CONSPIRACY	[{'span_text...	WyJUSElTIiwg...	218	37
1387	" I 'm deepl...	CRITICAL	[{'span_text...	WyJcdTIwMWMi...	294	48
13116	2021 : They ...	CRITICAL	[{'span_text...	WyIyMDIxIiwg...	198	28
11439	Anthony Fauc...	CRITICAL	[{'span_text...	WyJBbnRob255...	326	47
98	Proof has em...	CRITICAL	[{'span_text...	WyJQcm9vZiIs...	698	105

Figure 1: Structure of the original data, first 5 rows in DataFrame listed.

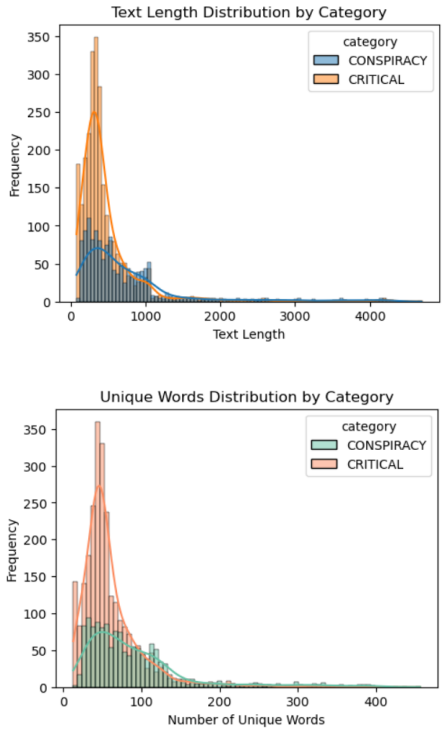

Figure 2: Text data distribution based on text length and number of unique words in each text.

Word clouds in NLP visually represents the frequency of words in a text, which helps quickly identifying the most important terms and themes within a dataset. The most frequent words are *vaccine, people, COVID*, appearing larger and more prominent in both categories, whereas *Deep State, NWO, New World Order, Ukraine* has an inclination towards conspiracy and *Pfizer, FDA, injuries* towards critical.

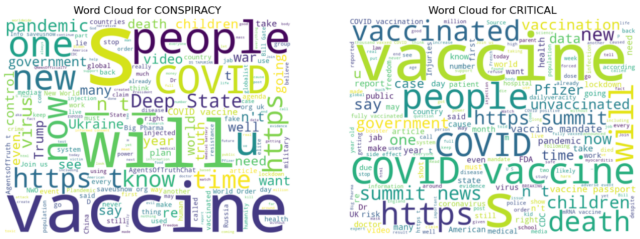

Figure 3: Word clouds over each category.

The reason for integrating out `annotations` column might be it is already well performed only applying word embedding on the tokenized data, but it still worths looking into these features. Thus, the annotations are expanded as below Fig.4 and have 7 categories Fig.5 for deeper insights

span_text	category	annotator	start_char	end_char
Austral...	CAMPAIGNER	gold_label	16	50
the fir...	CAMPAIGNER	gold_label	135	155
I 'm de...	NEGATIV...	gold_label	2	135
to vacc...	OBJECTIVE	gold_label	38	65
these c...	VICTIM	gold_label	51	65

Figure 4: Information of the annotations.

	category	count
0	CAMPAIGNER	5096
1	AGENT	5082
2	NEGATIVE_EFFECT	4387
3	VICTIM	3517
4	FACILITATOR	2763
5	OBJECTIVE	1602
6	X	206

Figure 5: Categories of the annotations

3.2 Pre-processing

Since tokenized data is already given, the pre-processing tasks left are normalization. In this project, I remove the punctuation, stop words using `stopwords` set in `nltk` lib and extract stems with `PorterStemmer` method also provided by `nltk` based on the tokenized texts. So the first thing is to decode the spacy tokens from `utf-8` to words.

Those methods are implemented in simple functions. The generated results are shown below. Stemming is used as an enhancement for stop words removal instead of a solo one.

decoded_tokens	process_stpw_stem	process_stpw
[THIS, IS, MASSIVE, Australian, Senator, Malco...	thi is massiv australian senat malcolm robert ...	this is massive australian senator malcolm rob...
[', I, 'm, deeply, concerned, that, the, push,...	i deepli concern push vaccin children noth dys...	i deeply concerned push vaccinate children not...

Figure 6: Pre-processing result examples.

Before step further into vectorization, I manage to explore other visualization tool using the cleaned data. One way is to degrade the dimension of the the text corpus based on distance and map it into a low dimensional space [1]. The text data needs to be first interpreted as vectors using TF-IDF and then put into T-SNE for dimension reduction to show how text data scattered in the 3D space. The results are not perceptible but still look nice at least.

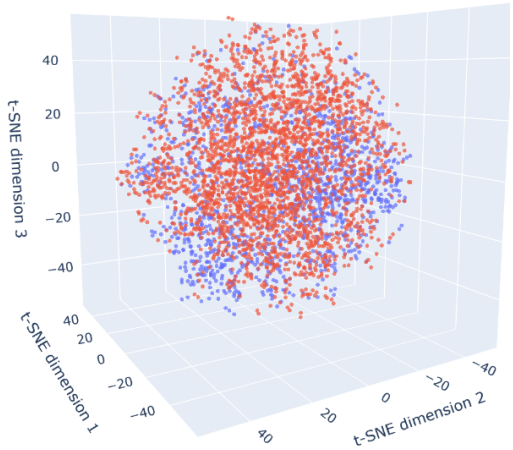

Figure 7: Dimension reduction using T-SNE.

3.3 Text classification

The main tasks for this section are:

(1) Split the dataset with 20% text data.
(2) Compare the performance of Naive Bayes model (annotated as NB) and feed forward neural network (annotated as FNN). NB can be implemented with the classifier `MultinomialNB` in `sklearn` and FNN with `Keras in TensorFlow`.
(3) Compare the performance of difference pre-processing pipelines and vectorization methods. 2 pipelines are `stop words removal` and `stop words removal with stemming`, which can be directly implemented with the function provided in the previous task. Vectorization methods are count vectorizer and TF-IDF. And then implement them in the 2 models and then do the comparison.

All the tasks are down in one class. Metrics for final evaluations are stored in a dict, which are updated respectively from NB model and the epoch history of FNN

```
class ModelComparison
```

Comparison Results

The synthetic results are shown in Table.1, that using counter vectorizer/BoW improve the accuracy significantly for naive bayes classifier but not necessary for FNN, since FNN is evidently reaching close-to-optimal result. Stemming is a good enhancement for normalization, showing that pre-processing pipeline should be considered in NLP tasks instead of just using one method. More illustrative results can be seen below.

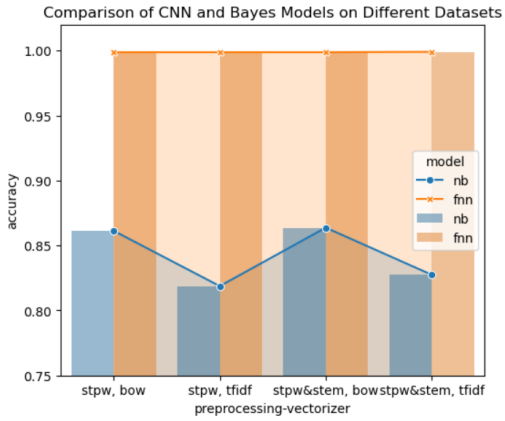

Figure 8: Comparison across different methods and models.

Specificly, the best results for NB is using pre-processing pipline and TFI-DF vectorizer, the results report is in Table.2 and plotted below.

Table 2: Report: NB, stpw&stem, TFI-DF.

	precision	recall	f1-score	support
CONSPIRACY	0.959677	0.472222	0.632979	252.0
CRITICAL	0.803254	0.990876	0.887255	548.0

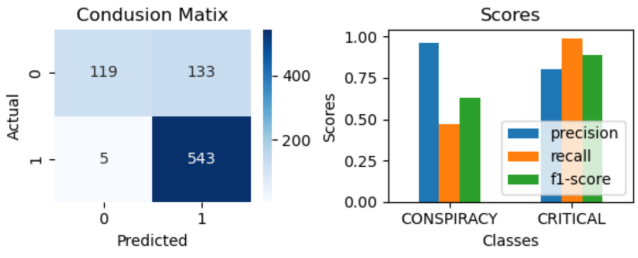

Figure 9: Confusion matrix: NB, stpw&stem, TFI-DF. Report scores: NB, stpw&stem, TFI-DF.

Here I implement a 3-layer FNN with Keras, parameterized as below:

(1) Optimizer: Adam;
(2) Regularization: dropout;
(3) Loss function: cross entropy;
(4) Evaluation score: f1, accuracy

layer	size	activation
1 fully connected	128	ReLU
2 fully connected	64	ReLU
3 output	2	SoftMax

The network is stable along the training procedure as shown in Fig.10 with close-to-optimal performance. The best epoch is consistently around the 6th epoch.

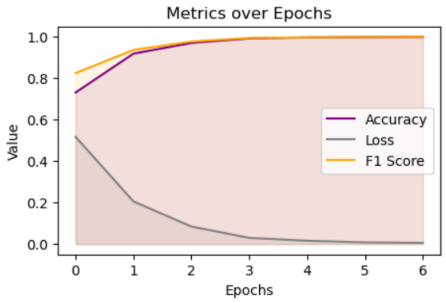

Figure 10: FNN results: stpw&stem, BoW.

3.4 PMI based word similarity

In this section, the discussions is about providing the most similar words based on the computed word-word pointwise mutual information matrix. Without using available packages, the PMI could be implemented as followed:

(1) Count the co-occurrence of each word with its context words with window size of 2 (required).
(2) Calculate the PMI matrix with minimum co-occurrence count of 2, which is set to avoid None value in the results. Find target words with PMI matrices. Then highlight the 10 randomly chosen terms and the most similar words.

	Random Word	Similar Word 1	Similar Word 2	Similar Word 3	Similar Word 4	Similar Word 5
0	indigen	drank	indigen	pair	rcmp	owner
1	67287290	wild	viru	case	caus	vaccin
2	kmoral	alison34	dm	scan	locat	pressur
3	comic	infin	hela	catherineholt	unforeseen	marvel
4	confidenti	usd	domain	rapper	artist	ema
5	macron	bastil	apec	touquet	cordon	9416347
6	côte	drc	algerian	cameroon	morocco	resid
7	far	reevalu	mathematician	privera	fetch	extream
8	africa	600ad	leon	liberia	bloodi	inund
9	haha	siiiy	ackshual	fiu	alzheim	prevent

Figure 11: PMI results - generated with 10 random words.

Table 1: Comparison of Models, Vectorizers, and Processing

	0	1	2	3	4	5	6	7
model	NB	NB	NB	NB	FNN	FNN	FNN	FNN
vectorizer	BoW	tfidf	BoW	TF-IDF	BoW	TF-IDF	BoW	TF-IDF
accuracy	0.86125	0.81875	0.86375	0.8275	0.99875	0.99875	0.99875	0.999062
process	stpw	stpw	stpw&stem	stpw&stem	stpw	stpw	stpw&stem	stpw&stem

4 Conclusion

The initial data analysis over text length, unique word count distribution, and word clouds are practical for understanding the data structure. For the NB model, using TF-IDF vectorization combined with stop word removal and stemming preprocessing pipeline significantly improved classification accuracy, achieving near-optimal results. Specifically, the NB model with TF-IDF vectorization demonstrated high precision and recall for the "conspiracy" category. FNN models showed stable performance under the same preprocessing and vectorization conditions, particularly after parameter optimization.

In conclusion, selecting appropriate preprocessing methods and vectorization techniques are crucial for improving text classification accuracy. The NB model particularly benefits from TF-IDF vectorization and stemming preprocessing, while the FNN model demonstrates high stability and accuracy after parameter optimization.

References

[1] Jp Hwang. 30/03/2020. NLP visualizations for clear, immediate insights into text data and outputs. *Plotly* (30/03/2020). https://medium.com/plotly/nlp-visualisations-for-clear-immediate-insights-into-text-data-and-outputs-9ebfab168d5b