CS 7311 Project Report
Texas State University
Fall 2020

Maria Tomasso
met48@txstate.edu

Andrew Magill
andrewmagill@txstate.edu

1. Introduction

Demand for efficient methods of detecting fake news has risen as social media becomes a primary source of news for a larger proportion of adults. In fact, in 2018 Pew Research found that twenty percent of adults in the United States cite social media as their primary source of news [7]. The sharing of fake news and conspiracy theories on social media can therefore misinform a significant number of people and lead to real world consequences. For example, in 2017 Allcott and Gentzkow estimated that U.S. adults were exposed to an average of "one to several" fake news articles through social media in the lead-up to the 2016 election and that pro-Trump fake news was both more common and more shared than pro-Clinton fake news [1]. While it is difficult to quantify the true impact of fake news on the 2016 election, these startling statistics emphasize the urgent need for more moderation on social media. Due to the massive volume of data produced by sites such as Twitter and Facebook, manual moderation is not a viable solution and automatic detection methods are needed. In this paper, we explore two strategies for detecting tweets promoting conspiracy theories linking 5G broadband cellular networks with the ongoing COVID-19 pandemic.

2. Problem Statement

Fake news and misinformation can spread rapidly on social media, especially during times of crisis. Many conspiracy theories about the coronavirus have circulated online since the initial outbreak in December 2019. Misinformation can affect compliance with public health measures such as mask-wearing and stay-at-home orders. Due to the volume of data created on social media, human moderators alone cannot control the propagation of fake news. In this paper, our goal is to develop models that can identify tweets promoting 5G/COVID-19 conspiracy theories using textual and structural data.

2.1 Text based analysis of tweets

The text-based classification task aims to classify tweets based on their content. Due to the nature of the platform, the style of language used in tweets may differ quite a bit from the corpora to which text classification models are often applied. Twitter has long been described as a micro-blogging social network, micro because each tweet is limited to 240 characters. Due to this restriction, tweets may often contain language that very concisely expresses meaning, including colloquialisms, abbreviations, hashtags, emoticons, and media. Tweets are often written in an informal fashion, and may contain spelling and grammar errors (tweets are not editable) more frequently than other text.

The uniqueness of Twitter content makes for a challenging text-classification problem. We attempt to exploit this characteristic of the corpus by taking a lexical rather than semantic approach. With such rich content, meaning may be difficult to infer, we instead attempt to identify differences in style exhibited by patterns of lexical usage.

We extend our approach in two ways, by augmenting our text data with information gleaned from associated media, and attempting to include the activities of users outside of our data set to make predictions based on inferred community membership. We use optical character recognition to mine images contained in tweets, and infer community membership from a much larger set of tweets and user interactions.

2.2 Structure based analysis of tweets

The goal of the structure-based classification task is to identify COVID-19 conspiracy tweets based on propagation patterns alone.

Each graph corresponds to a single tweet. Vertices represent accounts that retweeted it. Edges represent followership. Graphs are labeled as '5G-corona-conspiracy', 'non-conspiracy', or 'other-conspiracy'. There are: 2327 total graphs, with 270 in the '5G-corona-conspiracy' category, 1660 in the 'non-conspiracy' category, and 397 in the 'other-conspiracy' category. For each graph we are given 'nodes.csv', 'edges.txt', and 'plot.png'. We can disregard the plot, and work only with the edges and nodes files.

3. Related Work

3.1 Text based analysis of tweets

Recent work has employed logistic regression to classify tweets based on topic [4]. We have found that this approach works well with some modifications. We rely on the assumption described by Zhou et al. that writing style can be identified using lexical analysis by looking at frequency of word usage [8]. We analyze processing techniques that best differentiate the style unique to each class.

3.2 Structure based analysis of tweets

Recent literature has established that fake news propagates differently than real news on social media websites such as Twitter [6]. Although more research has been done on text-based classification of news, structure-based classification has the potential to provide an effective classification method while bypassing the challenges involved in natural language processing.

While the difference in propagation patterns between fake news and trusted news has been well documented ([6]), little work has been done on whole-graph classification. Monti, et al successfully trained a convolutional neural network to classify URLs as fake or trusted news based on their propagation patterns, however due to differences in the data structure their approach was not directly applicable to this problem [5].

4. Methods

4.1 Text based analysis of tweets

We construct a pipeline capable of learning and predicting fine and coarse grained classes in our data. We are provided with a list of tweets for each class, the content of which we must retrieve using the TwitterAPI, which provides content as well as associated media, user identification, and other information. We ingest the text content, user identity, and associated media for use at different points in our pipeline, which is comprised of six steps: normalize, tokenize, vectorize, model, prediction, and evaluation.

We may refer to the normalization step as preprocessing, this is the step where we make adjustments to our data in the hopes of increasing the predictive power of the content. This is the closest we get to analyzing meaning, we want to map non-identical tokens that share meaning to a uniform encoding. We judge the efficacy of our preprocessing steps by the change produced in our prediction evaluation. We discovered that our preprocessing techniques rarely resulted in improvements in both precision and recall, with each step we have to balance the benefit of normalizing our content with the potential loss of distinctiveness of each class.

Of the preprocessing techniques we attempted, we found the best results from transforming text to lowercase, removing punctuation, preserving URLs, removing stopwords, and normalizing terms ('u.k.' to 'uk'). Some of

the methods that did not benefit our predictions included preserving and isolating emoticons, and stemming or lemmatizing our text.

After normalizing the text, we move to the tokenization step. Here we split our content into individual terms. We experimented with tokenization patterns, in conjunction with the normalization step, to find the tokenization method that worked best. We attempted to split terms at whitespace and at punctuation, to include emoticons and to remove them. The results were surprising, the best method used a pattern that selected as tokens connected alpha-numeric characters, splitting at special characters and eliminating punctuation and emoticons all together.

Figure 1. The data pipeline for the text-based classification task.

Before learning from our data we must transform it into a numerical representation usable by our model. This is the feature extraction step. Common text to vector feature extraction representation include *Bag-of-Words*, Term Frequency-Inverse Word Frequency, and word embeddings. The *Bag-of-Words* technique, the simplest, is accomplished by constructing a vocabulary of terms and generating a vector for each data representing term occurence in the content. This method produced the best results for our dataset.

We fed our feature vector to eight different classifiers in order to select the best model for our data, these included: AdaBoost, Decision Tree, Gradient Boosting, Linear SVC, Logistic Regression, Multi-Layer Perceptron, Multinomial Naive Bayes, Random Forest, Stochastic Gradient Descent, and C-Support Vector Classification. We found Logistic Regression and the MLP Classifier consistently produced the best predictions, with the former just edging out the latter, and much more efficiently.

We evaluated our predictions against ground truth and used three scores to judge the goodness of our prediction: precision, recall, and Matthew's Correlation Coefficient. Precision is calculated as the true positives over true and false positives, recall is the true positives over true positives and false negative, and Matthew's Correlation Coefficient is a function of both. All scores have a maximum value of 1.0.

4.2 Structure based analysis of tweets

Due to the relatively small sample size and lack of uniform dimensions when the graphs were represented as adjacency matrices, a feature vector was extracted from each graph for use in classification. Feature vector extraction was done in R using the 'igraph' package. The feature vector consisted of nineteen metrics: number of nodes, number of edges, diameter of the graph, mean distance, edge density, reciprocity, global transitivity, local transitivity, number of triangles, mean in-degree, maximum in-degree, minimum in-degree, mean out-degree, maximum out-degree, minimum in-degree, mean total degree, maximum total degree, and minimum total degree. Metrics in the feature vector were selected from summary statistics for graphs given by prominent graph data repositories including SNAP, Koblenz, and UC Irvine. Each vector was assigned two labels: one identifying its class as given in the training data ('5G-conspiracy', 'other conspiracy', 'non-conspiracy') and one identifying it as '5G-conspiracy' or 'non-5G conspiracy' for the coarse classifier. After preparing the data, the resulting data frame was saved to a CSV and the rest of the analysis was performed in Python. An 80/20 train/test split was applied before three different types of models were trained. These models were:

- Decision tree with no maximum depth and the Gini impurity criterion
- Linear discriminant analysis (LDA) with SVD, LSQR, and LSQR plus shrinkage
- Naive Bayes

Each type of model was evaluated on both the coarse data ('5G-conspiracy', 'non-5G conspiracy') and the multi-class data ('5G-conspiracy', 'other conspiracy', 'non-conspiracy'). An additional outcome of 'cannot be determined' was added to both sets of outcomes. All models were trained in Python using the package scikit-learn and all features were standardized prior to training. After the model training, a principal components analysis was performed in an attempt to achieve better separation by label. PCA was also done in scikit-learn. Finally, two deep learning models were trained using StellarGraph. The architecture for both models was based off of examples in the StellarGraph documentation [2] [3].

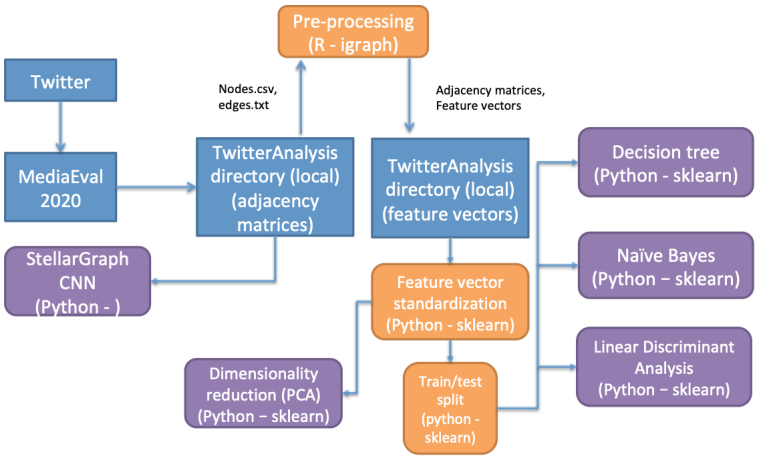


Figure 2. The data pipeline for the structural branch of this project.

5. Tools and Infrastructure

5.1 Text based analysis of tweets

Our pipeline was written in the Python language, version 3.7, and executed interactively in the web browser using Jupyter Notebook version 6.1.5. All data was stored locally in text JSON and CSV formats. We used Pandas and NumPy for data manipulation, NLTK for NLP tasks involving normalization methods, such as stemming and lemmatization and pytesseract for optical character recognition.

5.2 Structure based analysis of tweets

All coding was accomplished in either R or Python. Data preprocessing was accomplished in R version 4.0.2 with RStudio version 1.2.1335. Model training and assessment was done in Python 3.7.3 with Jupyter Notebooks version 5.7.8. All dataframes, both input and output, were stored locally as CSVs.

6. Data Processing Tools

6.1. Machine Learning Tools

6.1.1 Text based analysis of tweets

We made great use of machine learning utilities and modules provided by Scikit-learn version 0.23.2. We chained our processes together using the pipeline module, used the feature extraction module for tokenization and vectorization, the metrics module for evaluations, model selection module for train and test split, and the ensemble, linear model, Naive Bayes, Neural Network, and SVM modules for machine learning.

6.1.2 Structure based analysis of tweets

Scikit-learn (0.20.3) and pandas (0.24.2) were the primary packages used in this analysis. Pandas was used for dataframe management and scikit-learn was used to train and evaluate the decision trees, naive Bayes models, LDA models, and PCA. The deep learning models also used scikit-learn, as well as StellarGraph (1.2.1) and TensorFlow (2.1.0).

6.2. Visualization Tools

6.2.1 Text based analysis of tweets

We used Matplotlib version 3.3.2 to produce bar charts of evaluation scores, and Pandas version 1.1.3 to generate tables in Jupyter Notebook.

6.2.2 Structure based analysis of tweets

Visualization for all plots in the structure-based sub-task was done using Seaborn (0.9.0).

7. Experiments

7.1 Text based analysis of tweets

Average performance of coarse-grained classification over five runs with random 80/20 splits of the data into train and test sets show best performance from Logistic Regression with the MLP Classifier a close second.

Baseline Performance

model	mcc	precision	recall
AdaBoost-SAMME	0.36	0.61	0.33
Decision Tree	0.36	0.51	0.43
Gradient Boosting	0.35	0.73	0.24
Linear SVC	0.42	0.56	**0.49**
Logistic Regression	**0.43**	0.63	0.42
MLP Classifier	0.43	0.58	0.47
Multinomial NB	0.33	0.70	0.22
Random Forest	0.29	**0.91**	0.11
SGD	0.41	0.55	0.47
SVC	0.36	0.86	0.19

Figure 3. Performance of the baseline text based classification pipeline.

Applying all the net positive pre-processing techniques, we have increased our predictive performance MCC: +0.034, PREC: +0.009, REC: +0.032.

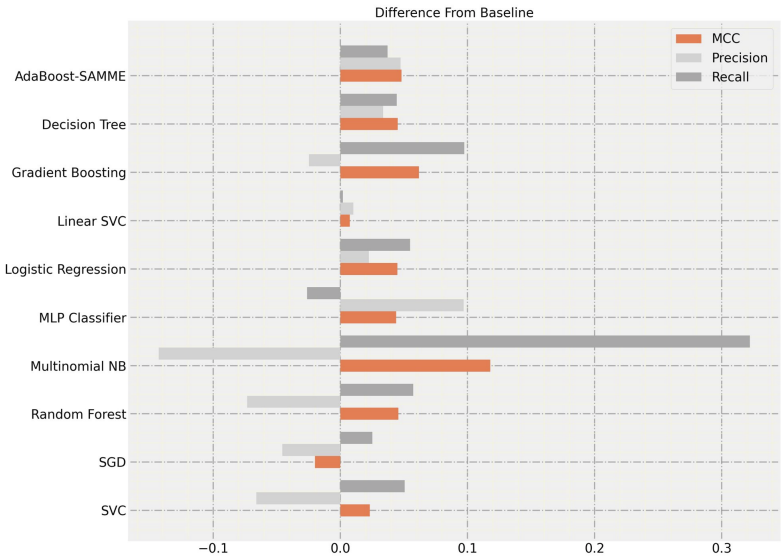


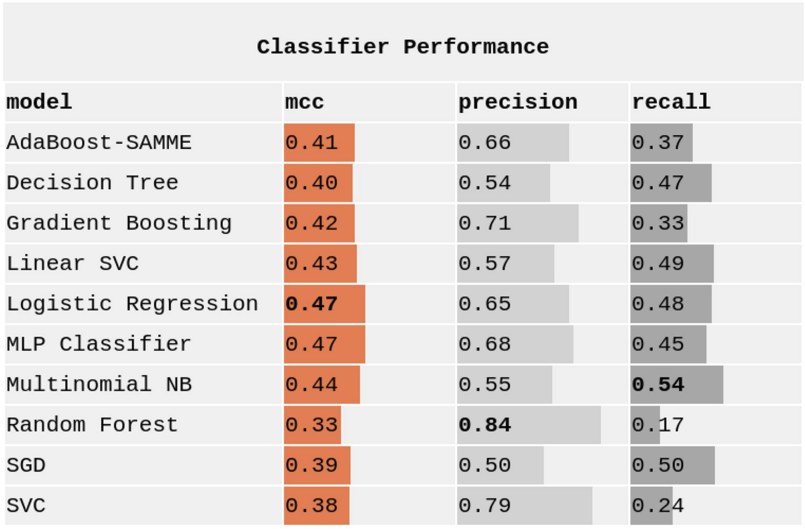
Figure 4. Improvements made over the baseline.

After iterating through classification parameters to find the best combination for our data. Our optimizations and preprocessing together produce improvements over the baseline with Logistic Regression continuing to outperform the other classifiers.

Classifier Performance

model	mcc	precision	recall
AdaBoost-SAMME	0.32	0.59	0.45
Decision Tree	0.31	0.51	0.49
Gradient Boosting	0.37	0.56	0.47
Linear SVC	0.37	0.54	0.52
Logistic Regression	**0.42**	0.62	0.50
MLP Classifier	0.42	0.58	0.53
Multinomial NB	0.39	0.55	**0.54**
Random Forest	0.34	**0.75**	0.44
SGD	0.34	0.51	0.51
SVC	0.36	0.59	0.45

Figure 5. Final results for fine-grained classification



Classifier Performance

model	mcc	precision	recall
AdaBoost-SAMME	0.41	0.66	0.37
Decision Tree	0.40	0.54	0.47
Gradient Boosting	0.42	0.71	0.33
Linear SVC	0.43	0.57	0.49
Logistic Regression	**0.47**	0.65	0.48
MLP Classifier	0.47	0.68	0.45
Multinomial NB	0.44	0.55	**0.54**
Random Forest	0.33	**0.84**	0.17
SGD	0.39	0.50	0.50
SVC	0.38	0.79	0.24

Figure 6. Final results for coarse-grained classification

It is interesting to note that our best classifier's precision has decreased while recall and MCC have increased. It seems that it is easier to attain high precision that high recall.

7.2 Structure based analysis of tweets

After training all models, the Matthew's correlation coefficient (MCC), accuracy, precision (macro), and recall (macro) were computed for the test data.

Naive Bayes

Models were trained on all three labels for a total of three runs. Results are summarized in the table below.

Model	Class	MCC	Accuracy	Precision	Recall
Naïve Bayes	multi	0.176	0.691	0.469	0.423
Naïve Bayes	coarse (i)	0.158	0.859	0.581	0.576
Naïve Bayes	coarse (ii)	0.126	0.709	0.591	0.544

Linear discriminant analysis

Three solving methods were tested in the LDA models. They were: (i) singular value decomposition; (ii) least squares; and (iii) least squares with shrinkage. Models were trained on all three labels for a total of nine runs. Results are summarized in the table below.

Model	Class	MCC	Accuracy	Precision	Recall
LDA SVD	multi	0.060	0.707	0.387	0.356
LDA LSQR without shrinkage	multi	0.049	0.702	0.375	0.354
LDA LSQR with shrinkage	multi	0.057	0.711	0.398	0.351
LDA SVD	coarse (i)	0.070	0.896	0.579	0.516
LDA LSQR without shrinkage	coarse (i)	0.070	0.896	0.579	0.516
LDA LSQR with shrinkage	coarse (i)	0.037	0.898	0.553	0.507
LDA SVD	coarse (ii)	0.173	0.722	0.624	0.560
LDA LSQR without shrinkage	coarse (ii)	0.167	0.720	0.619	0.558
LDA LSQR with shrinkage	coarse (ii)	0.140	0.722	0.620	0.541

Decision tree

In the decision tree models, the criterion and max depth hyper-parameters were tuned. Criterion could be set to either 'Gini' or 'entropy' and the max depth was either not set or set to three. Models were trained on all three labels for a total of twelve runs. Results are summarized in the table below.

Model	Class	MCC	Accuracy	Precision	Recall
Decision Tree, gini, no max depth	multi	0.048	0.572	0.356	0.356
Decision Tree, entropy, no max depth	multi	0.029	0.567	0.347	0.350
Decision Tree, gini, max depth of 3	multi	-0.002	0.711	0.278	0.331
Decision Tree, entropy, max depth of 3	multi	-0.002	0.711	0.278	0.331
Decision Tree, gini, no max depth	coarse (i)	0.015	0.809	0.507	0.508
Decision Tree, entropy, no max depth	coarse (i)	0.017	0.811	0.508	0.509
Decision Tree, gini, max depth of 3	coarse (i)	-0.026	0.898	0.452	0.496
Decision Tree, entropy, max depth of 3	coarse (i)	0.000	0.904	0.452	0.500
Decision Tree, gini, no max depth	coarse (ii)	0.127	0.648	0.563	0.564
Decision Tree, entropy, no max depth	coarse (ii)	0.158	0.665	0.580	0.578
Decision Tree, gini, max depth of 3	coarse (ii)	0.159	0.713	0.607	0.559
Decision Tree, entropy, max depth of 3	coarse (ii)	0.176	0.715	0.615	0.567

Principal components analysis

A principal components analysis was performed to see if a clearer separation of labels was possible. All variables were standardized to mean 0 and standard deviation one before PCA. With 5 components, 86.456 percent of variability was explained. The variability by component is summarized below.

Component	Variability explained
1	39.226
2	22.392
3	11.185
4	7.323
5	6.330

Unfortunately, separation was not seen for any labeling scheme when plotting by the first and second components.

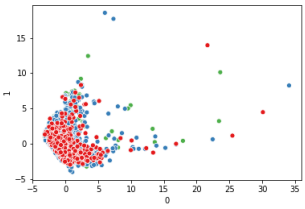


Figure 7. multi-class

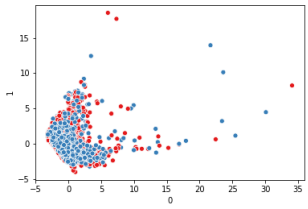


Figure 8. coarse (i)

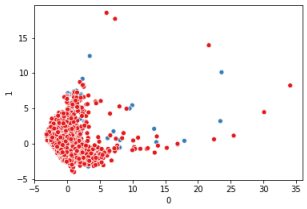


Figure 9. coarse (ii)

Deep learning

StellarGraph is a deep learning library that was recently developed for graphs. The documentation had two examples of neural networks being trained for graph classification, and we adapted the examples for use on our data [2] [3]. Unfortunately, the models were unsuccessful and gave each graph the same label. This failure was likely due to our data set being much too small to properly train a neural network.

The most successful model in terms of the MCC for both the multi-class and coarse tasks was the Naive Bayes model, but the MCC values were still disappointingly low. An analysis of why this approach failed can be found in the Conclusion.

8. Conclusion

8.0.1 Text based analysis of tweets

We were surprised to find that none of our modifications improved predictive performance across the board for all classifiers We found that some normalization was helpful, converting text to lowercase, for example, but it was important to retain the defining characteristics of each class We conclude that the most important steps we took were those that reduced noise, for instance, removing stopwords that are not likely to be predictive, and as it turns out, emojis.

8.0.2 Structure based analysis of tweets

Unfortunately, despite promising results in the literature we were unable to train a successful structure-based classification model for the identification of COVID-19 5G conspiracy related tweets. We believe our model failed to meet expectations for three reasons:

- The successful models in the literature focused on linked news articles, while our dataset was comprised of text-only tweets.
- While the literature has demonstrated a difference in propagation patterns between conspiracy and non-conspiracy tweets, it has not been established that COVID-19 5G conspiracy theories are shared differently than other conspiracy related tweets.
- The feature vector was selected based on the summary statistics given by leading graph repositories, not from the literature. We have not found any other papers that have tried this feature vector approach to graph classification, so we started from scratch in defining the feature vector. It is possible that, with more refinement, this approach could yield better results.

References

[1] H. Allcott and M. Gentzkow. Social media and fake news in the 2016 election. *Journal of Economic Perspectives*, 31:211–236, 2017.

[2] S. Documentation. Supervised graph classification with deep graph cnn. *https://stellargraph.readthedocs.io/en/stable/demos/graph-classification/dgcnn-graph-classification.html*.

[3] S. Documentation. Supervised graph classification with gcn. *https://stellargraph.readthedocs.io/en/stable/demos/graph-classification/gcn-supervised-graph-classification.html*.

[4] I. et al. Using logistic regression method to classify tweets into the selected topics. In *Intl. Conf. on Advanced Computer Science and Information Systems (ICACSIS)*, page 385–390, NY, Oct 2016. IEEE.

[5] M. et al. Fake news detection on social media using geometric deep learning, 2019.

[6] V. et al. The spread of true and false news online. *Science*, 359(6380):1146–1151, 2018.

[7] E. Shearer. Social media outpaces print newspapers in the u.s. as a news source. *https://www.pewresearch.org/fact-tank/2018/12/10/social-media-outpaces-print-newspapers-in-the-u-s-as-a-news-source/*, 2018.

[8] Zhou and Zafarani. A survey of fake news: Fundamental theories, detection methods, and opportunities. *ACM Comput. Surv.*, 53(5), Sept. 2020.