Getting Real on Fake News

Connor Masini and Thanaphong Phongpreecha

I. Problem Description

Recently, the circulation and distribution of news articles of dubious credibility has become very widespread. With the ease of sharing pieces of news through social media platforms like Twitter and Facebook. These "Fake News" pieces can quickly spread and manipulate hundreds or thousands of users into thinking they are true. The purpose of this project is to create a tool for the analysis of the credibility of Twitter messages. Such a tool could allow users to check the validity of a piece of news they find questionable. To check the validity of the article, we will examine factors including follower count, following count, verified user account information, number of re-tweets, number of favorites, pace at which the piece of news spreads, and of course the information found in the message itself. The combination of all these factors should allow us to create an effective tool that determines the legitimacy of a news article.

- **Linguistic**: Analysis of the article headlines, contents, or users' stance using natural language processing (NLP) is one of the primary methods of analyzing credibility. The underlying assumption for this methodology is that since fake news pieces are intentionally created for financial or political gain rather than to report objective claims, they often contain opinionated and inflammatory language, crafted as clickbait or designed to incite confusion. [1] Using methods to capture linguistic features such as N-grams, bag-of-words, Latent Dirichlet Allocation (LDA) models, or word embedding with models such as recurrent neural networks (RNN) or supported vector machines (SVM), a number of studies have constructed models with varying success for different datasets. [2]–[6] These approaches, however, are limited by the fact that the linguistic characteristics of fake news are not yet fully understood. Furthermore, the distinguishing characteristics of unreliable news vary across different topics, and media platforms. [7]
- **Network and Content Propagation:** It can be very challenging, if not impossible, to identify useful features from textural content alone as intentional spreaders of fake news may intentionally manipulate the content to make it appear like real news. To address this problem several studies instead focus on exploring the topology of information diffusion, *i.e.* graph mining. By employing RNN to capture features such as news propagation frequency combined with embedded user techniques, studies were able to achieve high F_1 and accuracy scores. [6], [8], [9]
- **Combinatorial:** Only a few recent studies have started to combine these features in the hope of achieving better accuracy within a shorter period of time (since we want to inhibit the propagation of false news). For example, a model integrating user's features with the post's textural features and propagation resulted in a high accuracy of 90% successful classification by using Long-Short Term Memory (LSTM), a specific type of RNN. [7], [10]. In a separate study, that used just the user and textural features with a logistic regression classifier, a lower classification success rate was achieved. [11] This emphasizes the importance of selecting features as well as the efficacy of the RNN.

II. Background in Recurrent Neural Networks

Recurrent Neural Networks (RNNs) have shown great success in processing data sequences in application such as speech recognition, natural language pocessing,and language translation. A simple RNN model is usually expressed using following equations:

$$h_t = \sigma(W_{hx}x_t + W_{hh}h_{t-1} + b_h)$$
$$y_t = W_{hy}h_t + b_y,$$

where W_{hx}, W_{hh}, W_{hy}, b_h, and b_y are a set of weights and biases for inputs, hidden states, and outputs with an activation function σ to perform non linear function. However, it is well-known in the literature that simple RNN suffers from vanishing/exploding gradient problems, *i.e.* back propagation through time within the model loses information between cells that are far apart from each other. To address this problem, one of the solutions that was introduced is a long-short term memory (LSTM) RNN, in which cell memory and more gates were added to each cell. The mathematical detail of these gates and memory retention mechanism is well-established in the literature. [10] In short, it can be laid out as:

$$i_t = \sigma_{in}(W_i x_t + U_i h_{t-1} + b_i)$$
$$f_t = \sigma_{in}(W_f x_t + U_f h_{t-1} + b_f)$$
$$o_t = \sigma_{in}(W_o x_t + U_o h_{t-1} + b_o)$$
$$\tilde{c}_t = \sigma(W_o x_t + U_c h_{t-1} + b_c)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$
$$h_t = o_t \odot \sigma(c_t)$$

where σ_{in} is called inner activation (logistic) function which is bounded between 0 and 1, and $\odot$ denotes point-wise multiplication. The output layer of the LSTM model is

usually chosen to be as a linear map, *i.e.*,

$$y_t = W_{hy} h_t + b_y$$

This is the formulation of the original LSTM, where essentially on top of the hidden state (h_t) that RNN has, i_t represent an information gate of how information flow into the LSTM cell memory (c_t). On contrary, (f_t) represents a forget gate or what information should the cell memory discard. Lastly, the output gate of LSTM (o_t) uses a combination of inputs, previous hidden states. Therefore unlike simple RNN, the hidden state of LSTM is affected not only by input, but also the cell memory. All of these help retain the memory, or the gradient, which solve the previous simple RNN issue. However, these added gates also increased the number of parameters, and hence the expense of the calculation. Variants of LSTM to reduce these parameters by modifying gates were introduced. [12]–[14] However, for the sake of simplicity, this study will use the original LSTM as the model.

III. Objectives

Previous research has provided useful insights to tackle this issue, however, the models used in many studies also come with significant setbacks for use in real-world applications. For example, using content propagation would require the news to spread significantly before it can be detected and dealt with accordingly. Additionally, there is still a significant accuracy gap that can be improved.

In this work, we aim to classify fake news using a combination of linguistic and network features. Moreover, we aim to explore different models and their structures to improve the accuracy. For example, exploring the use of end-to-end memory network compared to either LSTM or GRU would allow us to create a more effective tool. End-to-end memory networks were shown to be able to capture long-term structure within sequences and are an effective way to classify documents. [15], [16] We would also explore the use of algorithms to capture textural features prior to feeding to the model, for example, instead of just bag-of-word or N-grams, we will be investigating word-embedding techniques such as word2vec that can better preserve meaning and relations between words. [17] We will be using data obtained through the Twitter API with labeled training data provided by a previous study. [1] [9]

1. http://alt.qcri.org/∼wgao/data/rumdect.zip

IV. Resources Used

A. Work Environment

For this project, we used used Google Colab, a cloud based Jupyter notebook for machine learning and data analysis. This allows us to work concurrently without the hassle of a version control utility like git or mercurial.

B. Packages Used

To perform the necessary data analysis and machine learning for this project, we used using the pandas, numpy, datetime, pickle, and sklearn libraries. All figures were produced using the matplotlib and seaborn packages and the Twitter API was accessed using the tweepy library.

V. Dataset

A. Twitter API

To gather data for this project, we accessed the Twitter API using the python tweepy library[2]. Using a dataset of tweet id's along with the corresponding topic of the tweet and truthfulness of said topic found in [9], we used the Twitter API to gather additional information about the tweet. Using this dataset in conjunction with the Twitter API, we were able to create a dataset consisting of over 217,000 data points pertaining to 549 stories, 381 of which contain false information and 178 of which contain trustworthy information. Since the set of labeled tweets we used originated from 2008, many of the tweets found in the dataset have been deleted. We discarded all deleted tweets from our dataset since the Twitter API does not allow us to access any information about deleted tweets. The Twitter API allows us to access numerical, textural, and time series data for each news topic. Numerical data consists primarily of information about the user who wrote the tweet in question. This includes things like the number of the users friends, the number of followers the user has, how many tweets the user has "favorited", or the number of times the user has tweeted. The textural data consists of the text sent out in the tweet and is a string of words with a maximum length of 140 characters. The Twitter API allows us to access the time at which a tweet was posted, allowing us to evaluate the propagation of each piece of news over time. These numerical, textural and time-series aspects of the data taken together make up the dataset we used to train our model.

2. http://www.tweepy.org/

B. Statistical Description of the Dataset

The overview of the amount of tweets with corresponding users and time span for true and false news are shown in Table I. The original number of tweets was 250,000, but after removing deleted tweets, we were left with 217,009 tweets. Of tweets, true article topics makes up about one-third of the data; however, the number of true tweets accounts for up almost three-quarter of the data.

Looking further into the statistics of the users in the dataset (Fig. II), we can observe a very diverse set of numerical features, which could cause a potentially heavy skew. More on this topic will be discussed in the pre-processing section. For the statistics of words in the dataset, Table III shows the top 25 word frequency of true and fake tweets. It should be noted that this is biased because of the abundance of tweets found for some individual topics.

	Topics	Tweets	Users	Average Time Span
Total	549	217,009	98,940	333 Hours
True	178	159,012	–	–
Fake	381	55,051	–	–

TABLE I: Overview of the elementary statistics of the dataset.

	favorite counts	followers counts	friends counts	statuses counts
count	217,009	217,009	217,009	217,009
mean	1.38	4.40	2,930	120,519
std	50.00	830,116	12,900	233,0139
min	0	0	0	1
max	10,680	54,410,260	1,154,710	8,574,561

TABLE II: Statistics of the Twitter users.

C. Data Preprocessing and Analysis

After pulling information from Twitter related to the tweet ID, we have access to 20 additional features: account_date, coordinates, description, favorite_count, followers_count, following, friends_count, language, location, name, place, post_date, profile_image_url_https, retweet_count, statuses_count, text, time_zone, user_id, utc_offset, and verified. Of course, much of this data is unnecessary for the purposes of our classifier. After processing the data, we eliminate the following fields: coordinates, profile_image_url_https, time_zone, utc_offset, following, place, and verified. For all the different data types, we normalized the data on a per topic basis, as the number of tweets pertaining to each topic varied wildly. Since there was an equal number of valid and fake topics, normalization of valid and fake news topics wasn't necessary.

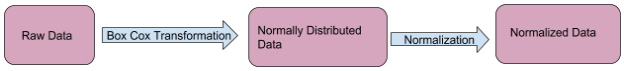

Fig. 1: Process diagram of the numerical feature pre-processing.

1) Numerical Data of Twitter Users: In its raw form, the numerical data associated with each tweet was not particularly useful. Each of the four analyzed variables was heavily skewed to the left, as many of the accounts did not have any favorites, friends or followers, but those that did could have thousands of each. A visualization of this raw data can be seen in Fig. 2. Heavily-skewed data can have an adverse effect to machine learning models, particularly artificial neural networks. To process the numerical data into a more useful form for our model, we follow the steps as shown in Fig. 2. First, we applied a Box Cox transformation to normalize the data. The Box Cox equation is effectively a class of log-transformation that can handle zeros and negative numbers. For our project, we only define Box Cox transformation for zeros and positive numbers, using the following formula:

$$f(x) = \begin{cases} \frac{y^{\lambda}-1}{\lambda}, & \text{if } \lambda \neq 0 \\ \log y, & \text{if } \lambda = 0 \end{cases}$$

False	Counts	True	Counts
starbucks	4,665	pool	23,091
cups	4,108	party	22,773
red	3,275	dead	16,658
woman	3,072	mckinney	16,525
king	3,066	rachel	16,437
burger	3,043	dolezal	16,009
christmas	2,945	dies	11,735
huggies	2,594	naacp	11,261
walmart	2,559	dat	11,076
giving	2,515	texas	10,829
new	2,469	bich	10,527
isis	2,450	video	10,342
wipes	2,377	weiland	9,758
glass	2,354	scott	9,515
found	2,312	man	8,982
dylann	2,311	phuc	8,743
obama	2,300	black	8,716
says	2,287	facebook	8,013
president	2,238	doritos	7,721
roof	2,204	police	7,644
baby	2,094	temple	7,030
trump	2,003	death	6,962
arrest	1,987	who	6,902
hot	1,926	pilots	6,819
holiday	1,924	officer	6,807
facebook	1,915	sandra	6,768

TABLE III: Top 25 stemmed words from the tweets classified as true and false.

All values of λ from [-5, 5] are considered and the optimal value for our data is selected. The optimal value is the one which results in the best approximation of a normal distribution curve.

The transformed data is shown in Fig. 3 with true and false news separately plotted. Compared to unprocessed data, friend counts, follower counts, and statuses counts have been successfully normalized, while favorite count was too skewed for Box Cox transformation. From the figure, we can see that there is not a significant difference between the numerical features of users who spread false and true articles.

To obtain more insight related to the numerical features and its correlation to veracity of the content, scatter plots between two numerical features with labeled data points were constructed. As shown in Figures 4, combining multiple numerical features together shows that there are sections of the correlations where true tweets were more concentrated, particularly the scatter plot between follower and friend counts. This motivated us to incorporate numerical features, including friends counts, follower counts, and statuses count as a part of an input to our model.

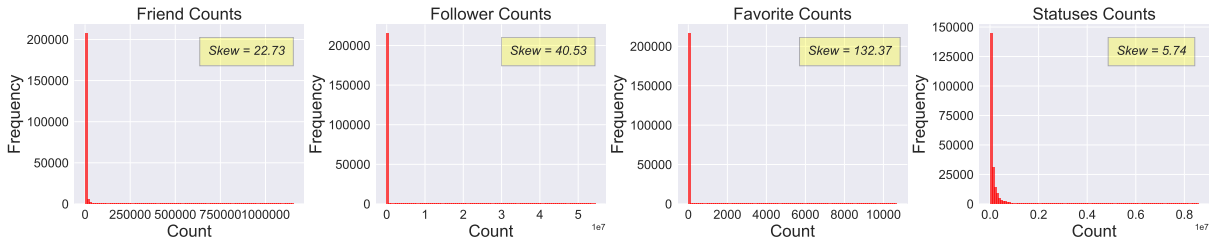

Fig. 2: Summary of the raw numerical features of Twitter users.

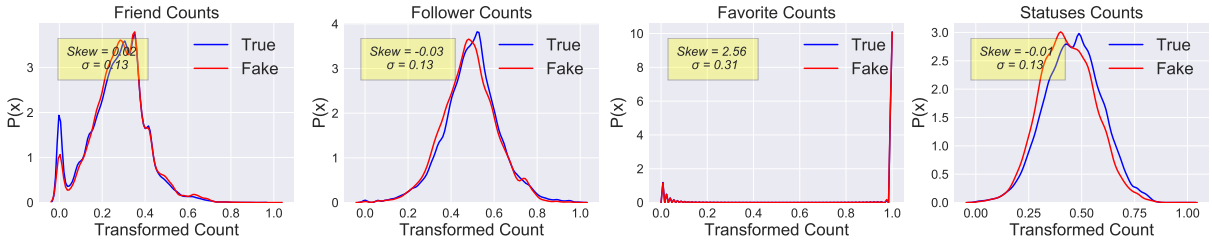

Fig. 3: Summary of the numerical features of tweets as transformed by two-parameter Box Cox transformation.

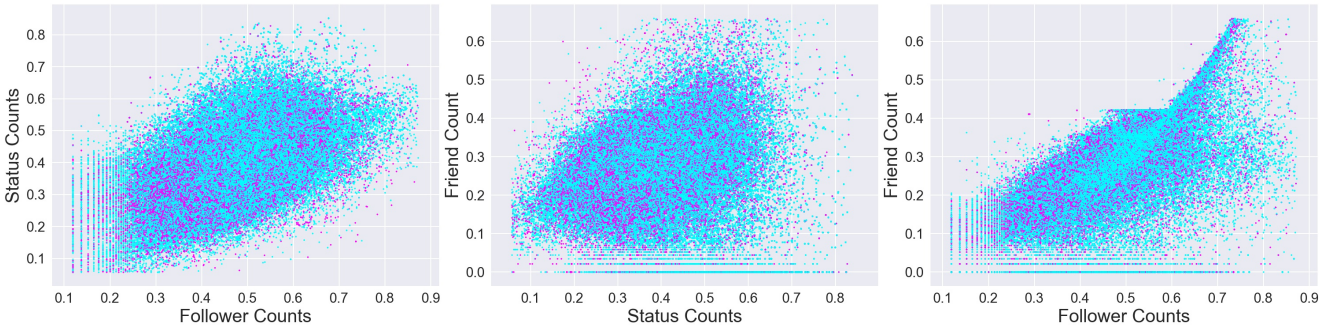

Fig. 4: Scatter plots of Twitter users numerical features with labeled true (blue) and false (pink) articles.

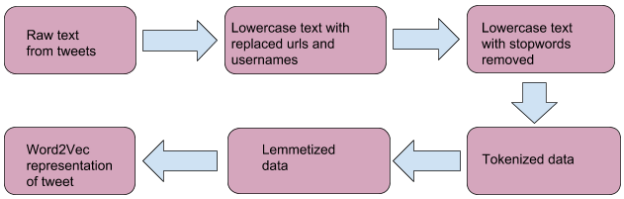

Fig. 5: Process diagram of the textural feature pre-processing.

2) Textural Data Preprocessing: Textural data in its raw form is not very useful for data analysis. Particularly for social media content, the textural data can contain a lot of noise – irrelevant data, such as urls, symbols, and high proportion of stop words to sentence lengths. Furthermore, to be an input to models, these data need to be converted to vectors. Therefore, we follow the steps outlined in Fig. 5 to tailor textural data to be more cohesive and eventually become representative vectors.

In the first step, we used regular expressions to replace words and patterns that are not useful for our study, including tagged usernames and source urls. After generalizing these specific parts of the tweets, we performed a 1-gram frequency analysis of the data to locate common words that introduce noise to the data. Fig. 6 shows the counts of the 30 most frequently occurring words in our dataset (not just the stemmed word). Based on this analysis, we can see that "url!" and "@user" are among the most frequently seen words in tweets. However, since commonly occuring words are not valuable for our classification models, we will perform noise reduction by removing some words, including "!url", "@user", and several other common articles of speech including "a", "an", and "the". We avoid removing too many words since tweets are already rather short.

Following this, we tokenize each tweet into individual words and punctuation marks and lemmetize these tokens. The lemmatization process transforms different tenses of the same words or words with identical meanings to a single word. This is done to reduce the complexity of the data and to allow for additional patterns in the data to be detected.

This data was then converted to a numerical representation using the gensim's word2vec library [18], which is at its core, a neural network that is trained on a corpus to yield vector outputs. Using embedded method allows us to retain the proximity of the words, *i.e.* the distance between similar words, such as between France and Paris, is closer when compared to less relevant words, like France and McDonald's.

The following tweets are examples of raw, unprocessed tweets with a label indicating their veracity preceding them:

(False) Sorry, Mark Zuckerberg is not giving $4.5 billion to 1,000 random Facebook users https://t.co/CTS9PoUnj3 #getsocial via mashable

(False) School principal bans Santa, Thanksgiving and Pledge of Allegiance - VIDEO: PTA president says principal going... https://t.co/9hrNEE0YyW

(True) Scott Weiland Reportedly Dead at 48 - Exclaim! https://t.co/dxT3fyqN31

After replacing all user-names and links, and converting all text to lower-case, the tweets are as follows:

(False) sorry , mark zuckerberg is not giving $4.5 billion to 1,000 random facebook users !url via mashable

(False) school principal bans santa , thanksgiving and pledge of allegiance - video : pta president says principal going ... !url

(True) scott weiland reportedly dead at 48 - exclaim ! !url

After tokenization and the removal of stopwords, the tweets take the following form:

(False)['sorry', ',', 'mark', 'zuckerberg', 'is', 'not', 'giving', '$4.5', 'billion', '1,000', 'random', 'facebook', 'users', 'via', 'mashable']

(False)['school', 'principal', 'bans', 'santa', ',', 'thanksgiving', 'and', 'pledge', 'of' 'allegiance', '-', 'video', ':', 'pta', 'president', 'says', 'principal', 'going', '...']

(True)['scott', 'weiland', 'reportedly', 'dead', 'at', '48', '-', 'exclaim', '!']

This resulting data is then trained by word2vec to convert the tokenized words into a numerical representation that can be used by our model.

Prior to this conversion, we performed some basic analysis on the textural meta-data. Figure 7 shows the distribution of the number of words in a tweet for both fake and true news. Though there is not a large difference, fake tweets do tend to be slightly longer than true tweets.

An analysis of the frequency of symbols used in tweets can be found in Fig. 8. This analysis showed that there is not a significant difference in the proportions of special symbols like punctuation marks or numbers between true and false tweets. Additionally, the proportion of numbers was essentially the same between true and false tweets.

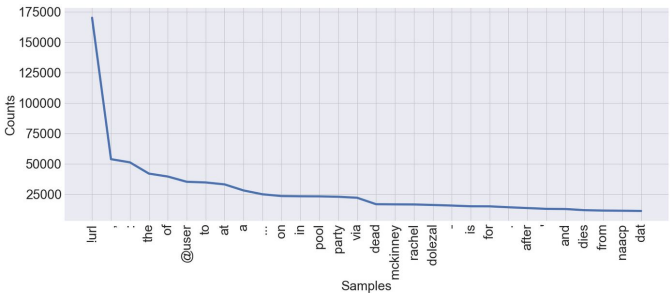

Fig. 6: Word frequency in the tweets.

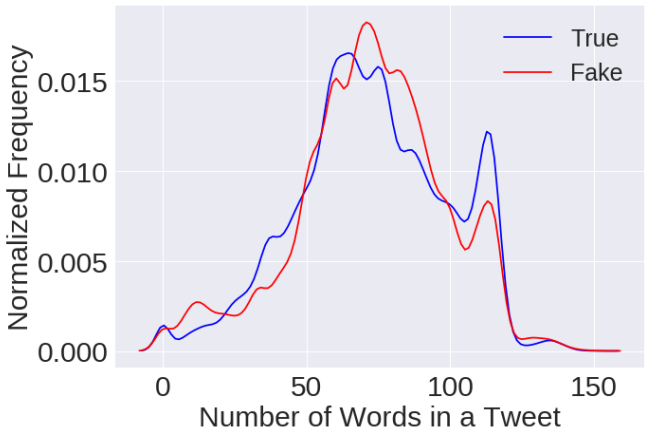

Fig. 7: Distribution of number of words per tweets in True and Fake tweets.

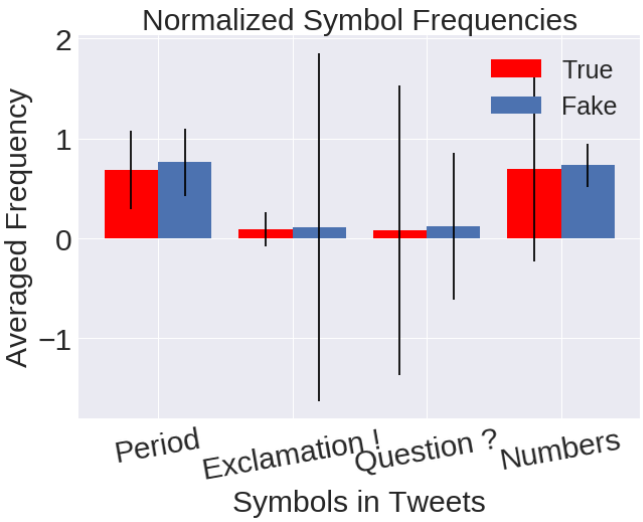

Fig. 8: Symbols used in Tweets

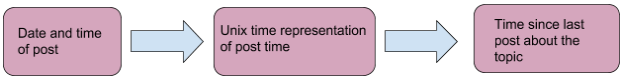

Fig. 9: Process diagram of the temporal feature pre-processing.

3) Time-Series Data Preprocessing: The API data from Twitter only provide time stamp of the post, and hence some pre-processing is required to convert them to period of time within each article instead. This is done by following the steps outlined in Fig. 9 For each tweet in our dataset, we used the Twitter API to determine the time and date at which that tweet was posted. We then converted the time at which the tweet was posted into seconds since January 1st, 1970 (commonly referred to as Unix time). We then found the time at which the first tweet for each topic was posted and found how long it took before other tweets about the same topic were posted. By analyzing this data, we saw that there is a difference in the propagation of true and false news pieces. As Fig. 10 and 11 demonstrate, real news tends to circulate slightly faster than fake news. This could be due to the circulation of real news by accounts with a high number of followers. These same accounts could recognize fake news as faulty and not share the misleading information, leading to a slower rate of circulation.

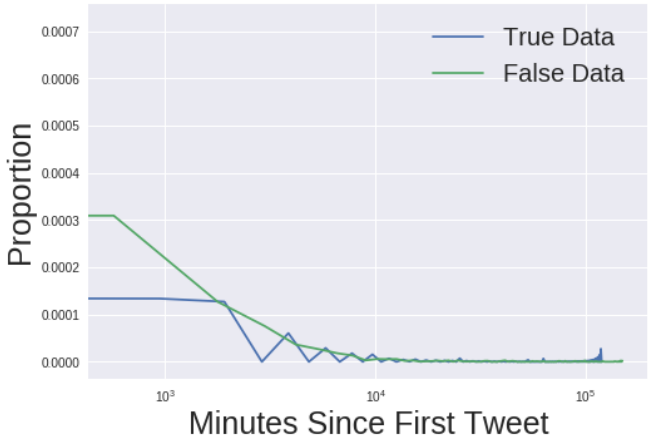

Fig. 10: Distribution of all fake and true tweets over time.

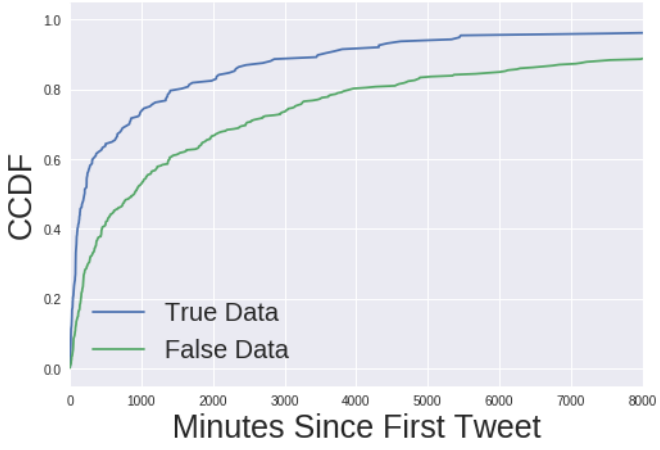

Fig. 11: Cumulative distribution of all fake and true tweets over time.

D. Conclusions from Data Analysis

From our preliminary data analysis, we learned that evaluating numerical data characteristics individually shows close to no correlation to whether the tweet is fake or true, but combining multiple numerical data features together may yield valuable information about the validity of a news source. Likewise, textural information about the data provides some valuable insight about the validity of the news piece in question. Of all the features investigated, temporal data shows the most promising results. This aligns with the findings found by a study performed earlier this year [19]. Overall, all of the different types of data showed promise for use in identifying fake news.

VI. Machine Learning Models

A. Inputs

The key component of the model input is the partitioning process and the choice of features used as inputs to the cells for each article.

Our feature vector x_t is a concatenation of several inputs with the following form:

$$x_t = (x_\tau, x_u, \Delta t, \eta),$$

where x_τ is the embedded textural content of the data, x_u is the data about user numerical features, including friend count, follower count, and status count. Δ_t and η are the temporal features of the data which are explained in further detail below.

Due to the sheer number of tweets in the dataset, it is not possible to use them in their raw data form. To shrink the number of tweets, we partitioned tweets in each article to different granularities. For example, for a partition of an hour, tweets that were posted within the same hour will then be combined together: the numerical features (x_u) of users within that partition will be averaged and the textural vector data (x_τ) of that partition is taken from the first tweet in that partition. The number of tweets within that partition is represented as η. Naturally, there will be partitions where there were no tweets (blank), in such case all of these features are zeros. Lastly, the Δt is the number of blank partitions between the current partition and the last non-blank partition.

The inputs x_t were also reshaped to have three dimensions – (number of article, number of partitions, number of features). The process is illustrated in Fig. 12.

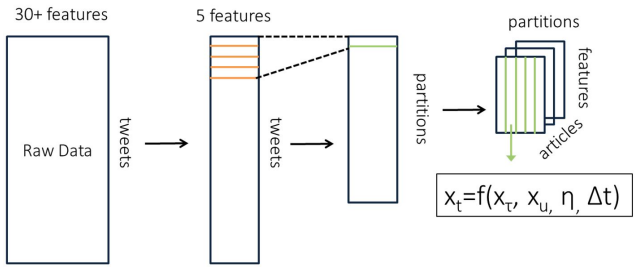

Fig. 12: Partitioning and reshaping of the input data for feeding into the neural network model.

B. Models

As the core of the module, we used a Recurrent Neural Network (RNN), since RNNs have been shown to be effective both at capturing temporal patterns in data, and for integrating different sources of information together. Specifically, to prevent the vanishing/exploding gradient issues in traditional RNNs, we employed long-short term memory (LSTM) RNN instead.

As the inputs are come from different domains, *i.e.* time series data, numerical data, and textural data, it is not desirable to incorporate them into the RNN as raw input. To standardize the input features, we insert an embedding layer between the raw features x_t and the inputs $\tilde{x}_t$ of the RNN. This embedding layer is a fully connected layer as following:

$$\tilde{x}_t = tanh\big(W_a x_t + b_a\big),$$

where W_a is a weight matrix applied to the raw features x_t at time partition t and b_a is the bias vector. The last hidden state h_T is then passed to the fully connected layer, resulting in a vector:

$$\tilde{v}_j = \sigma\big(W_r h_T + b_r\big),$$

where W_r is a weight matrix applied to the last hidden states, h_T, and v_j is the probability vector for the veracity of each story. Probabilities over .5 are then perceived as false news. The illustration of this concept and model is shown in Fig. 13.

The model is then trained using back-propagation in time to minimize the binary cross-entropy loss function defined as:

$$Loss = -\frac{1}{N}\sum_{j=1}^{N}\big[L_j \log \hat{L}_j + (1-L_j)\log(1-\hat{L}_j)\big] + \frac{\lambda}{2}||W||^2,$$

where L_j is the ground truth and $\hat{L}_j$ is the predicted value. The LSTM model is also subjected to $L2$-norm regularization and random dropout of the weights to minimize the effect from over-fitting. The model is then used for training over 5-fold cross validations.

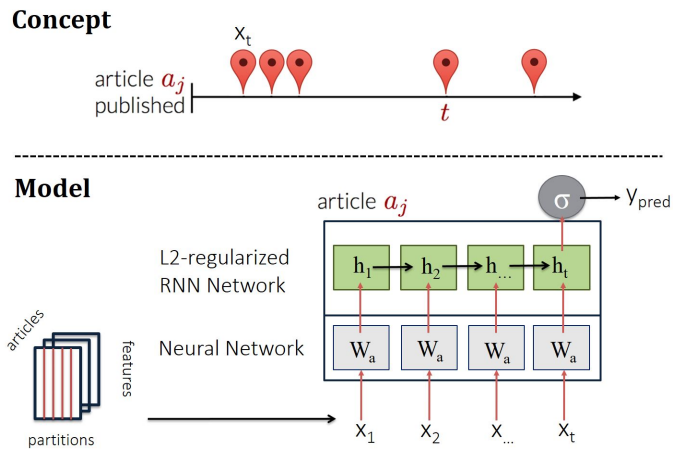

Fig. 13: Concept of the RNN as applied to each article and how it translates to RNN model.

VII. Results

A. Model Evaluation

The measurement of the model's performance based on the accuracy alone is not sufficient as this does not yield information regarding true negatives or false positives. In order to capture these aspects, two approaches (in addition to accuracy) will be considered in the following sections. These include:

1. The Receiver Operating Characteristic (ROC) and its area under the curve (AUC).

2. The F_1 score, which is a combination of both precision and recall. The formula for each of these measurements is:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

$$\text{precision} = \frac{TP}{TP + FP}$$

$$\text{recall} = \frac{TP}{TP + FN},$$

where TP, FP, and FN are true positives, false positives, and false negatives, respectively.

B. Model Performance in Different Time Span

This section will explore the change in performance of the model with a variable amount of partitions. The maximum time span of an article in the dataset is over 3.5 million minutes. Even with an hour granular per partition is applied, there will still be over 58,000 inputs to the model per article, which is very computationally expensive. On top of that, it is important to have a model that can capture false news as early as possible. Therefore, having a model with that achieves a high accuracy faster, *i.e.* using less partitions (and hence data points), is desirable.

To achieve this goal, a comparison overtime using 1-hour (60-minute) partition is investigated (Fig. 14 red). In this section the textural input is omitted to reduce the computation time as it is expected to have minimal effect. The results are shown in Fig. 14. From this figure, we can see that the highest accuracy and F_1 score for 1-hour partition of 0.72 and 0.75 (Fig. 14 top and bottom), respectively, were achieved when using partitions up to around only 8 hours.

Interestingly, using tweets from longer period of time (more 1-hour partitions) did not yield more accuracy as initially expected. On the contrary, the scores were reduced. This indicated that over the long run, the characteristics of the features between true and false news are similar, and hence harder to be separated.

The results from ROC and AUC do not corroborate completely the with accuracy and F_1 score of the model. The ROC result of 1-hour granularity is shown in Fig 15 top right. While the AUC of the model with inputs of 8 partitions, *i.e.* 8 hours, which exhibited the highest F_1 score, has one of the highest AUCs of 0.66, different numbers of maximum partition hours, like 2 or 20 hours, which did not show promising F_1 values, also had a comparable AUC to

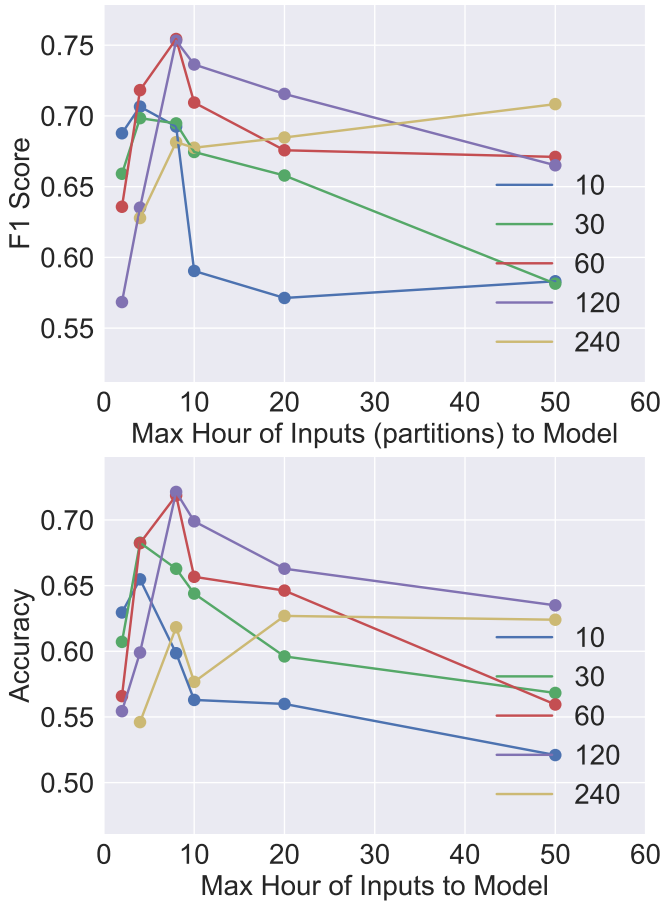

Fig. 14: The accuracy and F_1 score of the model using different granularities of partitions, including 10, 30, 60, 120, and 240 minutes per partition, with different maximum hours as an input to the model, including 2, 4, 8, 10, 20, and 50 maximum hours (this would translate to different number of partitions for different granularities)

the 8 hour one. Similarly, if we extend the hours of the 1-hour partition to 50 hours, the area dropped significantly to just 0.53.

C. Model Performance Using Different Granularity

Previously we have shown the maximum accuracy obtained by using 1-hour granular partition is by feeding RNN up to just 8 hour of the information. In this section, we will explore if the change in granularity from 1 hour to lower/higher, would enable us to achieve higher performance and its effect on where the highest performance is achieved. To do this, we investigated the granularity of 10 minutes, 30 minutes, 2 hours, and 4 hours and fed the model over different maximum periods of time. Similar to the previous section, the results are based on just the numerical and time-series input, no textural data is incorporated yet.

The F_1 and accuracy score in Fig. 14 suggested that using 1-hour and 2-hour per partition resulted in the highest scores. Similarly, the results also suggest that there is an optimal maximum hours to be fed to the model, both of which showed the best result when we only fed up to 8 hours (8 partitions and 16 partitions for 1-hour and 2-hour, respectively). For most granularities, the trend is similar in that the best maximum hours of the input is around 8 hour, while increasing the maximum hours beyond this resulted in decreased scores. It makes sense that as time goes on, the similarity between true and fake news could converge, but having all models perform the best at 8 hours implies the need to have a larger dataset to confirm this behavior.

It should be noted there is an exception to the trend in the 4-hour (240-minute) granularity (Fig. 14, where it maintains the high score even at 50 hours. Further tests were performed up to the maximum input time of 100 hours, *i.e.* 25 partitions, and found that the score only decreased from the 50-hour mark (result not shown).

One notable result that was shown by the ROC and AUC of different partition granularities in Figure 15 (purple) is that the best result from the 2-hour partition (at 2 hours, or 4 partitions fed to the model) yielded an AUC of 0.79, significantly higher than other partitions at different maximum hours, including 1-hour partition at 8-hour input, which previously shown similar F_1 and accuracy score (Fig. 14 red)

Similar to previously discussed with the 1-hour granular, most results suggested that 8 hour input yields one of the highest performances, but it was much less clear-cut which one is the best.

Conditions	F_1	Accuracy
60-min granule, 8 h max	0.63	0.64
60-min granule, 8 h max	0.62	0.62

TABLE IV: The model performance after adding textural features into the input partitions.

D. Incorporating Textural Features to the Model

After we have identified the best granularity partition and the maximum hours to be fed to the model using just numerical and temporal inputs. We next add the textural feature to each of these partitions. The results are shown in Table. IV. As can be seen, adding the textural features decreases the F_1 and accuracy score significantly from around 0.75 to now around 0.63. Therefore, textural feature may not be the best for this model. However, the success from temporal inputs suggest that other meta-features of the twitter should be considered, these are mentioned more in detail in the Future Work suction (Section IX).

VIII. Conclusions

In conclusion, our work used the labeled tweet dataset to analyze the characteristics of true and false news and developed a RNN model to classify these tweets.

The data analyses of users' numerical data (follower counts, friend count, and status count), tweets' textural data, and tweets' temporal patterns showed that there were no significant difference between the numerical and textural features between true and false news. This reflects the reality

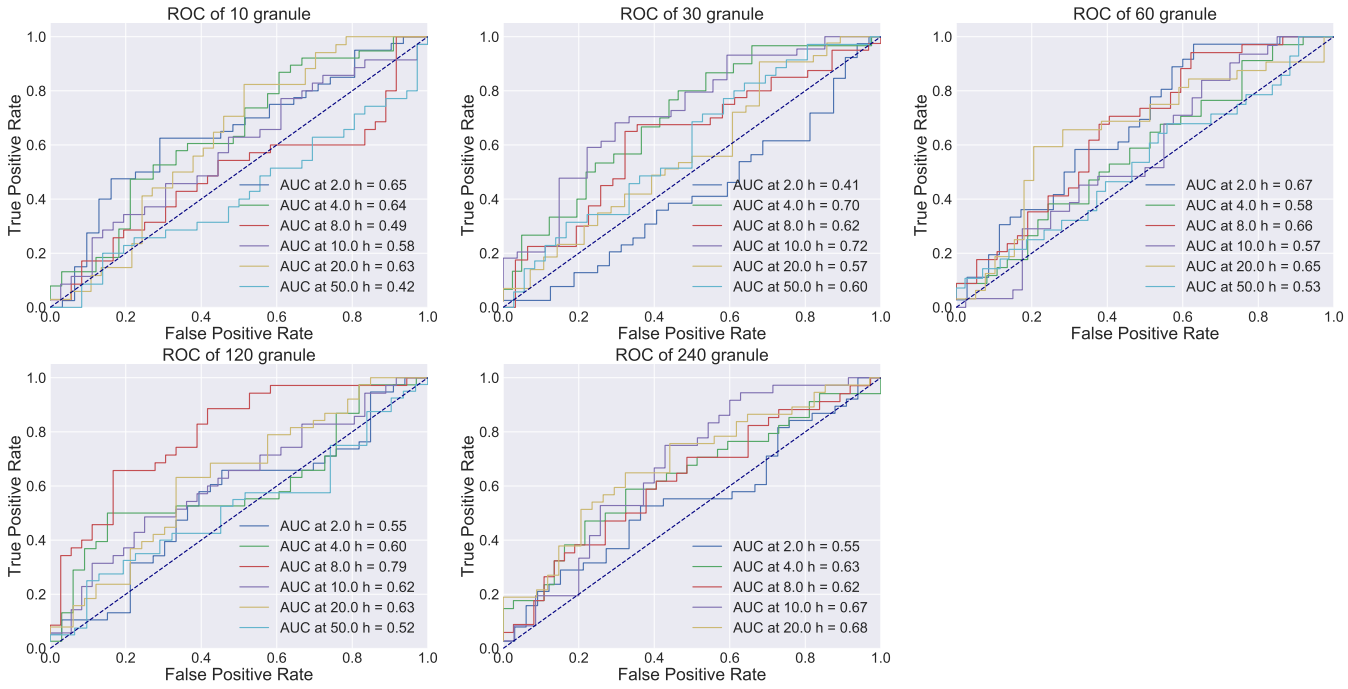

Fig. 15: The ROC and AUC of the model using different granularities of partitions, including 10, 30, 60, 120, and 240 minutes per partition, with different maximum hours as an input to the model, including 2, 4, 8, 10, 20, and 50 maximum hours (this would translate to different number of partitions for different granularities).

as many of these accounts intentionally create false content that is meant to appear real, or believe that the information they are sharing is in fact real. However, there is a significant difference in time series data, *i.e.* how information spread, as these are not controllable by the original author of the tweet.

Recurrent neural network models, specifically LSTM, were constructed to exploit the difference in temporal behavior between the true and false tweets. It was found that F_1 score and accuracy around 0.75 using the temporally partitioned numerical and temporal inputs to the model. Adding textural data into the input was found to decrease the performance down to around 0.63.

IX. Future Work

A. Graph Analysis

The rate of information spreading is the biggest driver to our model. Therefore, the direction of how it spread, *i.e.* interactions between users should be further explored and incorporated into the model, possibly by using a high rank sparse matrix to represent the interactions between users. These interactions could potentially provide valuable insight into which user accounts are more likely to tweet credible pieces of information.

B. Retweet Analysis

Another way in which this work could be expanded is to expand the analysis of numerical and time-series data to the retweets of the original tweet. A retweet is when another user chooses to share someone else's tweet with his or her followers. Analysis of the users who retweeted the original tweet and when they retweeted the piece of news could be done just as it was for users who posted tweets in our experiment. This would allow the creation of a larger training dataset, with more information about the propagation of a given tweet, presumably leading to a more accurate classification model.

Project Timeline

Week 1	2/19-2/25	• Literature Review • Dataset Acquisition • Set-up Development Environment
Weeks 2-4	2/26-3/18	• Design Models • Examine/Preprocess Features
Weeks 5-7	3/19-4/8	• Analyze Deep Features • Refining Architecture • Model Training
Weeks 8-9	4/9-4/25	• Finalize Results • Prepare Final Report • Project Submission

References

[1] Yimin Chen, Niall J Conroy, and Victoria L Rubin. Misleading online content: Recognizing clickbait as false news. In *Proceedings of the 2015 ACM on Workshop on Multimodal Deception Detection*, pages 15–19. ACM, 2015.

[2] David M Markowitz and Jeffrey T Hancock. Linguistic traces of a scientific fraud: The case of diederik stapel. *PloS one*, 9(8):e105937, 2014.

[3] Svitlana Volkova, Kyle Shaffer, Jin Yea Jang, and Nathan Hodas. Separating facts from fiction: Linguistic models to classify suspicious and trusted news posts on twitter. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 647–653, 2017.

[4] Verónica Pérez-Rosas, Bennett Kleinberg, Alexandra Lefevre, and Rada Mihalcea. Automatic detection of fake news. *arXiv preprint arXiv:1708.07104*, 2017.

[5] Aditi Gupta, Ponnurangam Kumaraguru, Carlos Castillo, and Patrick Meier. Tweetcred: Real-time credibility assessment of content on twitter. In *International Conference on Social Informatics*, pages 228–243. Springer, 2014.

[6] Jing Ma, Wei Gao, Zhongyu Wei, Yueming Lu, and Kam-Fai Wong. Detect rumors using time series of social context information on microblogging websites. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 1751–1754. ACM, 2015.

[7] Natali Ruchansky, Sungyong Seo, and Yan Liu. CSI: A Hybrid Deep Model for Fake News Detection. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 797–806. ACM, 2017.

[8] Liang Wu and Huan Liu. Tracing Fake-News Footprints: Characterizing Social Media Messages by How They Propagate. 2018.

[9] Jing Ma, Wei Gao, Prasenjit Mitra, Sejeong Kwon, Bernard J Jansen, Kam-Fai Wong, and Meeyoung Cha. Detecting rumors from microblogs with recurrent neural networks. In *IJCAI*, pages 3818–3824, 2016.

[10] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[11] Arkaitz Zubiaga. Learning class-specific word representations for early detection of hoaxes in social media. *arXiv preprint arXiv:1801.07311*, 2018.

[12] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2017.

[13] Felix A Gers and E Schmidhuber. Lstm recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340, 2001.

[14] Tsung-Hsien Wen, David Vandyke, Nikola Mrksic, Milica Gasic, Lina M Rojas-Barahona, Pei-Hao Su, Stefan Ultes, and Steve Young. A network-based end-to-end trainable task-oriented dialogue system. *arXiv preprint arXiv:1604.04562*, 2016.

[15] Zheng Li, Yu Zhang, Ying Wei, Yuxiang Wu, and Qiang Yang. End-to-end adversarial memory network for cross-domain sentiment classification. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, page 2237, 2017.

[16] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448, 2015.

[17] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International Conference on Machine Learning*, pages 1188–1196, 2014.

[18] R Rehurek and P Sojka. Gensim–python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, 3(2), 2011.

[19] Soroush Vosoughi, Deb Roy, and Sinan Aral. The spread of true and false news online. *Science*, 359(6380):1146–1151, 2018.