STA 561: Final Project

Jane Zhang, Avijit Mandal, Shyam Venkatasubramanian, Siddarth Madala

1 PR

Duke startup Flagger launches software product to identify spam and false information on social media platforms

Durham, NC—April 25, 2022—Today, Duke students Jane Zhang, Avijit Mandal, Shyam Venkatasubramanian, and Siddarth Madala launched Flagger—a developer API that helps social media platforms flag messages that contain spam and false information. The API ingests messages across different topics like politics, comedy, entertainment, and more and flags whether the message contains language conducive of spam or exaggerated claims using a in house machine learning model. The model was developed on data from mass social media platforms like Twitter and is intended to support moderation teams at social media companies in removing content that doesn't fit their terms of services.

"Discourse over media speech and censorship is more prevalent in our society than ever before. Just last week we saw Elon Musk propose to buy Twitter in attempt to allow complete free speech with support from his 82.5 million subscribers and in the last couple years debate over how much power to give social media platforms on removing harmful content has been highly debate in Section 230" explains Zhang who is pursuing a degree in computer science and statistics. "We believe technology and machine learning can be used to help facilitate as much free speech on social media platforms as possible while empowering social community moderators to filter harmful messages" says Mandal who is pursuing a degree in electrical engineering.

During the pandemic, on average people spent 1,300 hours on social media or about 15 percent of the year [2]. Social media has become an integral to communication and with companies like Meta and Snapchat to launching more virtual environments or the Metaverse, we will spend exponential more time online with access to anonymous communication. While this increases connectivity and accessibility of opportunities across the world, the increase in media usage is also conducive of more online misinformation and spam. Most recently, the rise of DAOs or decentralized autonomous organizations on the blockchain has yielded millions of dollars lost by people in anonymous communities through phishing messages around crypto currency. Thus, Flagger is embarking on a great challenge: automatically flag misinformation and spam before vulnerable users can be taken advantage of.

While big social media companies like Twitter and Meta may have vast engineering teams and moderation teams to fight spam, new communities like DAOs on Discord and emerging

social media platforms like Be Real and Clubhouse sacrifice moderation for initial growth, leading to media platforms like Parler which was taken advantage of for the January 6th US Capital attack. "That's where Flagger comes in. We have created an out of the box moderation API for any new social media platform to plug into their messaging system and prevent spam and harmful messages from being spread" says Venkatasubramanian who is pursuing a degree in electrical engineering.

2 FAQ

Technology and Data
Q1: How was the machine learning algorithm trained?
A1: Flagger will be built on a generative adversarial network (GAN) with a recurrent neural network (RNN) backbone. To determine the feasibility of this approach, we have developed a 'Generative RNN' framework consisting of an RNN for generating spam-like and not-spam-like messages, which is compared with a simple 'RNN' framework, an SVM framework, and a Logistic Regression framework. The dataset we have used for model training is the 'UtkML Twitter Spam Dataset', which necessitates sophisticated learning models for classifying tweets as spam or not-spam. Contrastingly, email spam and review spam datasets are easier to learn in general (the latter dataset is used in SpamGAN [1]). This is caused by the 240-character limitation placed by Twitter on each tweet which results in fewer words, making it more difficult to learn sequential dependencies between adjacent words. Regarding model training, we only consider the 'Tweet' column from the UtkML dataset, and we split it so that 80% is used for training and 20% is used for testing. Within the generative step, the RNN is trained using the Adam Optimizer with learning rate $\alpha = 1 \times 10^{-3}$, and within the classification step, the RNN is also trained using the Adam Optimizer with learning rate $\alpha = 1 \times 10^{-4}$. For additional specifics regarding the model architecture and dataset preprocessing, we refer the reader to section 3.

Q2: How was the machine learning algorithm evaluated?
A1: Our proposed 'Generative RNN' model is evaluated on the test examples from the 'UtkML Twitter Spam Dataset', and is compared with a simple 'RNN' framework, an SVM framework, and a Logistic Regression framework. The metric we have used is classification accuracy (what percentage of tweets are correctly classified as spam or not-spam), due to the balanced nature of the dataset - roughly 50% of the dataset consists of spam tweets and the remaining 50% consists of not-spam tweets.

Q3: How did you collect the training data and what does the data entail?
A3: Our training data for our initial model comes from publicly available pre-labeled spam datasets on Twitter tweets compiled by the University of Tennessee Machine Learning's group. You can access the data set in our repository: https://github.com/jjanezhang/final-project.

Q4: How does the Flagger API work?
A4: The Flagger API operates by allowing social media corporations to send methods through our systems via subscription and API key and receive a score indicating whether the post is likely to be spam or not. We allow our clients to use our API trained on existing Twitter data or to retrain our model using their own labeled data from their platform.

Q5: Are there limitations on the training input for the Flagger API?
A5: Clients are able to retrain the Flagger spam model using any text data they wish. We currently do not support images and videos or analysis of links in posts.

Q6: Are there limitations on the prediction output for the Flagger API?
A6: Our prediction outputs return confidence of spam as a percentage. We do not give any information on specific contents of the message that are considered spam.

Q7: How do you know if a message has false information in it?
A7: We have found through analysis over of more than 100,000 social media posts that spam messages with false information use language that is indicative of false information. Our model is trained based on those features; however, we do not have a solution implemented that corroborates accuracy of information. This is something we will look into moving forward.

Q8: What spam cases does the product best flag?
A8: We are better at identifying spam such as false advertisements and click bait messaging rather than true misinformation given that our model focuses on language heuristics rather than corroborating evidence.

Q9: What spam cases does the product struggle to flag?
A9: As of now, we have no services to flag videos or images. For text messages, we currently treat all links equally but recognize that often spam and misinformation is accompanied by phishing links. That being said, the introduction of video, image, and link analysis as well as finding solutions to cross check information accuracy is something we would like to look into moving forward.

Implementation

Q1: How much time and what resources do you need to implement the Flagger API?
A1: The Flagger API can be used out of the box by clients pre-trained on our Twitter data set. However, if the client chooses to retrain the model on their own data, they will have to estimate projected timeline for collecting their own data, labeling, and cleaning it. The model itself takes less than an hour to train.

Q2: How do existing social media platforms implement Flagger?

A2: Existing social media platforms such as Twitter and Facebook use their own, in-house spam services. These teams can use Flagger to compare results or spin up quick analysis on sample text corpuses while building out their own models.

Q3: Can social media platforms continuously train Flagger on their own data?
A3: Yes, existing teams can retrain the Flagger model using their own data. We provide documentation for how to do so and evaluate their models. Our development team is also ready to step in and work directly with the company team to optimize Flagger for the company's own data.

Q4 How can Flagger help teams with existing manual moderation teams?
A4: Flagger can aid or entirely replace manual moderation teams. Specifically, moderation teams can use the score from the Flagger model to help decide whether to label a post as spam. However, the company is also likely to set their own threshold for the Flagger confidence score to automatically label spam in their own data pipeline.

Q5: How can non technical team members understand the results from the Flagger API?
A5: Non-technical staff can see the confidence score regarding each post presented on an internal dashboard. We can also help build UI interfaces for teams to reject or approve of flagged messages. Thus, no technical background is necessary past the initial set up with the company's engineering team.

Impact and Privacy
Q1: How do you know you are increasing the safety of social media communities?
A1: We work closely with the teams using our API to see if outcomes have improved on their platforms. Specifically, we track metrics such as rate at which messages are flagged and find decreases in the rate to be indicators that we are helping to prevent misinformation and spam on the platform.

Q2: How do you protect user data privacy?
A2: User data is used in the retraining the model is only seen by the social media company's team unless released and shared with Flagger for QA. We do not store critical user information as we only take what is given to us in API calls from social media sites. In any case, we do not share or sell any user data without the permission of our clients on behalf of their users.

Q3: How do you protect data privacy across multiple clients?
A3: If we are allowed to use data from our clients for RD, we will not disclose results and efficacy of any of our client's info with other clients or prospective ones unless given permission.
Legal
Q1: Who assumes legal responsibility for the effects of falsely flagged messages?
A1: Flagger provides tools for social media companies to support their platform moderation

but we are not explicitly making content removal decisions for companies, and thus do not take responsibility for falsely flagged messages unless our metrics, but will take responsibility in the case where our scores are not producing as expected. In addition, we forego any responsibility for messages taken down mistakenly via decisions made of our services by citing Section 230 of Title 47 of the United States Communications Decency Act. We are neither a publisher nor speaker for any information provided by a user of or the information content provider themselves (i.e. social media services).

Future work
Q1: What problems are top of mind for you to work on in the next year?
A1: Currently, it is top of our mind are the expansion of our services to new small to medium social media platforms and communities rising. To better cater to these clients, we are focusing on building the easiest, out of box solution that can help abstract away moderation for all of these teams. This includes improving our model to consider images, videos, and link analysis.

Q2: How do you plan to expand the product in the next 5 years?
A2: We hope to develop solutions that corroborate misinformation beyond looking at language characteristics. We will also begin to think about solutions incorporating user data such as keeping a list of accounts that frequently post spam as well as link them via graph database to accounts believed to be run by the same person/entity across social media platforms. This will allow us to provide better services for all of our valued customers.

3 Technical Appendix and POC

3.1 RNN Background

As the RNN forms the backbone of our generative network, we introduce Recurrent Neural Networks in this section, deriving from [2]. Recurrent Neural Networks (RNN) are a family of neural networks that are used to process sequential data. The state of hidden units of a recurrent neural network h_t depends on the input at time x_t and the state or output at time $t-1$, h_{t-1}. The state h can be expressed as

$$h_t = f(x_t, h_{t-1}; \theta),$$

where θ is a parameter of the nonlinear function f. Figure 1 shows one unit in a recurrent network (depicted on the left side of the expression).

Recurrent networks are generally used in scenarios, where a data sample can be predicted using samples from the past. So, a recurrent network maps an arbitrary length of input $(x_t, x_{t-1}..., x_1)$ to h_t. The unfolded recurrence at time step t can be expressed as follows:

$$\begin{aligned} h_t &= f(h_t, x_t; \theta) \\ &= g_t(x_t, x_{t-1}..., x_1), \end{aligned}$$

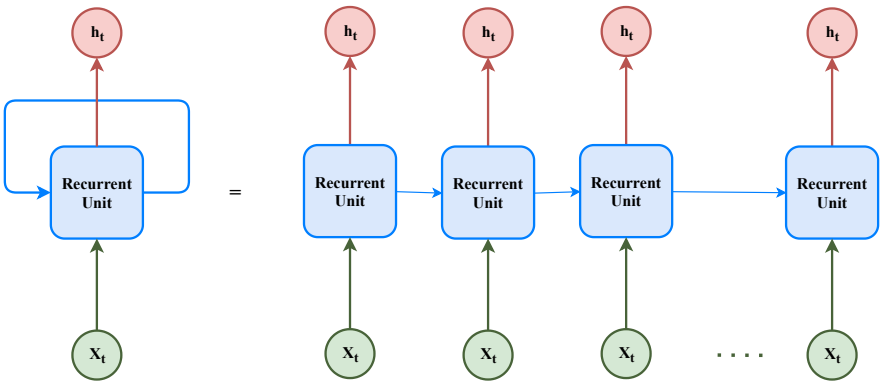

Figure 1: Recurrent network unit (left) and folded and unfolded recurrent network (right).

where g_t denotes repeated application of the function f. In Fig. 1, the unfolded version of a recurrent network unit has been shown. In Fig. 2, the overall structure of a recurrent network has been shown. The network maps an input sequence x to a corresponding sequence of outputs o. The loss L is computed to measure the error between o and a target sequence y. The network has input connections parameterized by a weight matrix U, and hidden-to-hidden recurrent connections parameterized by W, and hidden-to-output connections parameterized by V. In general, a RNN outputs a probability distribution for the next element x_t using current state h_t. The joint probability of all the elements can be decomposed as follows:

$$p(x_1, x_2, \ldots, x_t) = p(x_1)p(x_2|x_1)\cdots p(x_t|x_1, \ldots, x_{t-1}).$$

A RNN models conditional probability distribution using function g and h_t as

$$p(x_t|x_1, ..., x_{t-1}) = g(h_t).$$

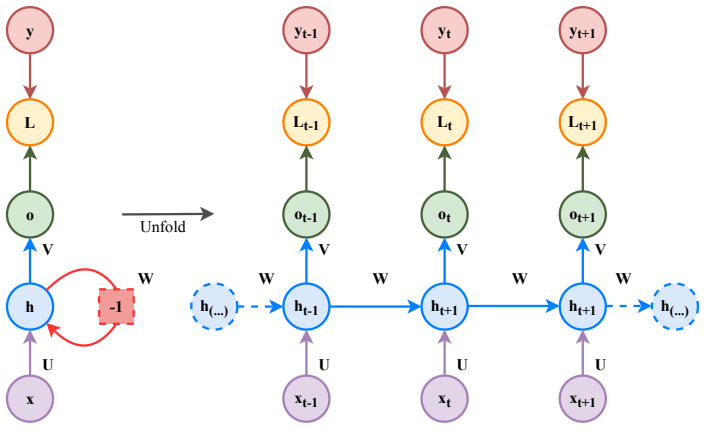

Figure 2: Overall recurrent neural network.

One problem with RNNs is that it often becomes difficult to train to capture long-term dependencies, because gradients tend to vanish. Even if the parameters of the recurrent

network are stable, the difficulty with long-term dependencies arise from the smaller weights given to long-term interactions (involving multiplications of many Jacobians) compared to short-term ones.

To handle this problem, gating units were proposed to be added to recurrent networks. One of the first attempts in this direction was the long short term memory (LSTM) unit. Recently, another recurrent unit called the gated recurrent unit (GRU) was proposed. As indicated above, these networks aim to capture long-term dependencies, which is unimportant for our considered dataset (due to the shortened nature of each tweet). Deriving from this notion, we only consider a simple RNN architecture in our Parallel Generative Network.

3.2 Generative RNN Framework

For our proof of concept, we propose a 'Generative RNN' deep learning pipeline to achieve improved spam detection accuracy over traditional learning models such as SVM and Logistic Regression. We will later extend this Generative RNN pipeline to a GAN framework akin to [1]. This Generative RNN consists of two parts: a parallel RNN architecture for generating spam-like and not-spam-like tweets, and an RNN which is used to classify the concatenated dataset consisting of generated and original tweets. For simplicity, the same neural network architecture is used for each RNN (spam and not-spam) in the parallel generative network. This network architecture is depicted below in Figure 3.

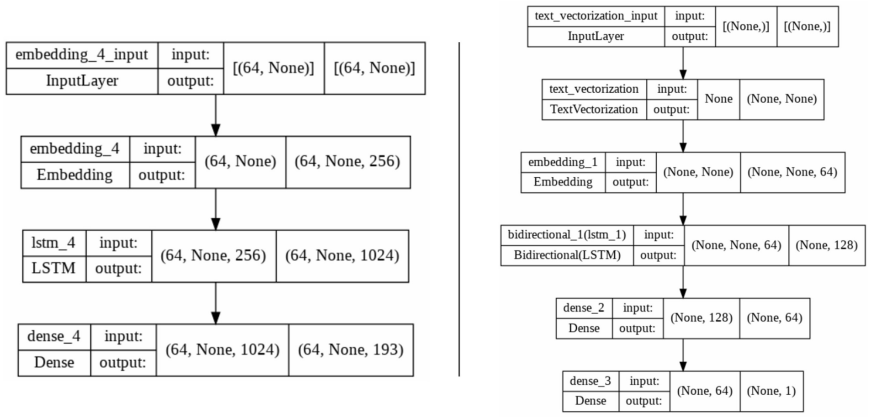

Figure 3: RNN from Parallel Generative Network (left), and RNN from Classification Network (right). The generated spam and not-spam tweets from (left) are concatenated with the original tweets in the dataset and fed into (right).

Regarding the parallel generative network, we train the RNNs on the spam and the not spam examples, respectively, from our original dataset for a period of 45 epochs (insufficient for convergence). To generate the individual spam-like and not-spam-like tweets, we input a string consisting of a single 'whitespace' into each RNN, which outputs a 240-character long 'generated tweet'. We expect the RNN trained on spam tweets to output spam-like tweets, where the distribution of the spam-like tweets closely resembles that of the spam tweets from

the original dataset. Similarly, we expect the RNN trained on not-spam tweets to output not-spam-like tweets, the distribution of which closely resembles that of the not-spam tweets in the original dataset.

To generate the spam-like and not-spam-like tweets, the RNN follows a character-based text generation procedure. As a first step, the dataset is split into the training and test datasets and preprocessed, whereupon all non-English characters are removed, as well as accented characters, special characters, stopwords, and numbers. The training dataset is then divided into spam tweets and not-spam tweets, where each tweet is padded with whitespace to reach 240 characters, after which each training dataset (spam and not spam) is flattened such that each tweet forms a sentence in a paragraph. Prior to this flattening step, we vectorize the tweets, whereupon each character from a given tweet (column index) is mapped to a numerical row index, forming a matrix. Each sentence can now be thought of as a training sequence. As a final step, we copy each sequence and move it to the right by a single character to form an 'input' sequence (e.g., 'This is a tweet lol') and a 'target' sequence (e.g., 'his is a tweet lolo'). The RNN will now try to use the characters from the input sequence to predict the next character, which will be compared to the last character in the target sequence. Training the Generative RNN model follows from this intuitive procedure and is outlined in our attached code. The predicted spam-like and not-spam-like sequences (tweets) outputted by the Parallel Generative Network are appended to the 'UtkML Twitter Spam Dataset', which is subsequently inputted into the RNN classifier.

3.3 Simulation Results

To characterize the performance of our 'Generative RNN' model on the 'UtkML Twitter Spam Dataset', we compare it with three other approaches: a simple 'RNN', an SVM framework, and a Logistic Regression framework. The comparison metric we have used is classification accuracy. To obtain the optimal hyperparameter combination needed to train our SVM and Logistic Regression models, we utilize an exhaustive K-fold Cross Validation procedure. This algorithm is depicted below:

Algorithm 1 K-Fold CV-based Hyperparameter Optimization through Exhaustive Search

1. Define $p = \{p_1, p_2, \dots, p_m\}$, $a = \{a_1, a_2, \dots, a_m\} = zeros(m, 1)$
2. **For** $i = 1, \dots, m$
3. **For** fold $k_i \in$ K
4. $X_{train} = X_{\{k \in K, k \neq k_i\}}$, $Y_{train} = Y_{\{k \in K, k \neq k_i\}}$, $X_{test} = X_{k_i}$, $Y_{test} = Y_{k_i}$
5. $Z.\text{init}(p) \rightarrow Z.\text{fit}(X_{train}, Y_{train})$
6. $a_i = Z.\text{score}(X_{test}, Y_{test}) + a_i$
7. **End**
8. $a_i = a_i / K$
9. **End**
10. $p_{opt} = p[\text{argmax}(a)]$

In the above pseudocode, $p \in \mathbf{R}^{m \times n}$ represents the set of all hyperparameter combinations, with $p_1,\ldots,p_m \in \mathbf{R}^n$, where p_i is the set of values taken by our n hyperparameters. Additionally, X is our training data set, Y contains our training labels, Z is our learning model (SVM or Logistic Regression), and $a \in R^m$ is the set of averaged model validation accuracies (through K-fold cross-validation) for our m hyperparameter combinations. Regarding the input dataset for the SVM, Logistic Regression, and simple RNN, 51.7% of the dataset consists of spam tweets and 48.3% consists of not-spam tweets. Contrastingly, for the Generative RNN case, the simple RNN classifier takes as input a concatenated dataset where 51.4% of the dataset consists of spam tweets (1250 spam and not-spam tweets [outputted by Parallel Generative Network] appended to original dataset). As an additional note, the original dataset undergoes the same preprocessing procedure that was implemented in the 'Generative RNN Framework'.

Cumulatively, our Generative RNN, RNN Classifier (Simple RNN), Logistic Regression Model, and SVM model are evaluated on the test dataset, which is randomly sampled from the original dataset via an 80/20 (training/testing) split. The classification accuracies of our selected models have been summarized below in Table 1; this accuracy is representative of the percentage of tweets that have been correctly classified as spam or not-spam.

Table 1: Classification Accuracy of Tested Learning Models

Learning Model	Classification Accuracy
RNN	86.97 %
Generative RNN	78.74 %
Logistic Regression	56.10 %
SVM	53.68 %

Evaluated on the test dataset, we observe that our RNN achieves the highest classification accuracy, followed by the Generative RNN. Contrastingly, the Logistic Regression and SVM models demonstrate nonexistent improvements over a random guessing approach. The reason why SVM and Logistic Regression perform far worse than our RNN models on this dataset can be attributed to their inability to learn any structure; the only information they retain are the words present across tweets and how frequently they appear. The RNN and Generative RNN, however, are able to directly learn phrases from tweets, and are able to form more complex text representations through the usage of word embeddings. Regarding why the simple RNN outperforms our Generative RNN, the latter model is not adaptive - there is no discriminator present to improve the spam-like and not-spam-like nature of tweets outputted by the Parallel Generative Network. Additionally, the short nature of most tweets makes it difficult for the Generative RNN to find patterns between adjacent words. Altogether, we would expect the accuracy of our proposed approach to improve (and likely best the RNN) were we to extend it to a GAN. Building this improved network is central to improving Flagger's deep learning architecture for spam detection.

References

[1] G. Stanton and A. A. Irissappane, "Gans for semi-supervised opinion spam detection," *CoRR*, vol. abs/1903.08289, 2019. [Online]. Available: http://arxiv.org/abs/1903.08289

[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.