Compromised Tweet Detection Using Siamese Networks and fastText Representations

Mihir Joshi
Faculty of computer science
Dalhousie University
Halifax, Canada
mihir@dal.ca

Parmeet Singh
Faculty of computer science
Dalhousie University
Halifax, Canada
pr819318@dal.ca

Nur Zincir-Heywood
Faculty of computer science
Dalhousie University
Halifax, Canada
zincir@cs.dal.ca

***Abstract*—The aim of this work is to detect compromised users of tweets based on their writing styles. In this paper, we use Siamese Networks to learn a representation of user tweets that allows us to classify them based on a limited amount of ground truth data. We propose the employment of this classification model to identify compromised user accounts of tweets.**

***Index Terms*—Cyber security, compromised users, author attribution, latent representation, siamese networks**

I. Introduction

With the advancement in network technologies and the vast amount of freely available resources on the internet, it is easier even for an unskilled person to hack someone's online account. Where an advanced attacker could use sophisticated resources like keyloggers, Remote Administration Tool (RAT) and Trojan horses, an inexperienced person could use phishing pages or various GUI based brute force tools readily available online. To this end, there are various instances of data breaches exposing millions of user accounts that are being sold on the black market. Although websites employ security measures to prevent unauthorized access, once the account credentials are acquired, the attacker can masquerade as the owner of the compromised account. The attacker would use the compromised account to send phishing links, spam, or fraudulent messages. Usually, these messages are short texts, i.e., 140 characters or fewer, especially on social media websites. These text messages also contain improvised vocabulary to make use of the character limit. Therefore, it is harder to find a pattern in these micro-messages compared to longer texts.

In text analysis, previous works [1, 2] aims to classify longer text corpora such as Reuters Corpus v.1 [3] while others [4, 5, 6] are designed for shorter texts, such as tweets. They use feature extraction and representation techniques, namely bag of words and word embeddings [7]. However, these techniques fail to learn latent representations from the text. Latent similarity between two documents is the semantic closeness between them based on the context. In this paper, we employed Siamese Networks [8] to explore the learning of latent representations for analyzing tweeted messages. As Horiguchi et al. [9] stated, latent representations or metric based features perform well when there is less data, as in the case here. Since there is a maximum of 100 tweets per user, the metric-based features, i.e., contrastive loss, performs better than the Softmax based ones.

The rest of the paper is organized as follows. Section II gives an overview of the related literature. Section III introduces the dataset and the feature extraction and representation techniques used as well as introduce the proposed approach. Section IV discusses the experiments and evaluation results of the proposed system. Finally, conclusions are drawn and future work is discussed in Section V.

II. Related Works

Research on classifying authors of short text messages based on their writing styles have been studied previously after the advent of social media websites like Twitter and Facebook. Layton et al. [10] carried out one of the earlier researches on the short text using Twitter data. They proposed the lazy learning approach using character n-grams. Their system calculated the most frequent character n-grams after combining all tweets for a single author and then compared it to a new data sample using the Simplified Profile Intersection (SPI) [11] similarity. Later, Schwartz et al. [4] extended the character n-gram approach when they proposed the supervised learning model using Support Vector Machines (SVM) [12]. Their system is also trained on the data collected from Twitter users and features are extracted using character n-grams, word n-grams, and flexible patterns [13].

Combining deep networks with supervised learning, Rhodes [14] implemented a model based on convolutional neural networks (CNN) using word embeddings [7]. They created word vectors that are trained using the skip-gram approach before feeding it to CNNs. Although their research was based on longer texts [15] it formed a basis of the system implemented by Shrestha et al. [5]. They adopted the similar CNN architecture but instead of using word vectors their system is based on the sequence of character n-grams. They constructed unigrams and bigrams from short text data [4] to build character embedding before applying CNNs.

Recently, there are several systems that implemented another form of artificial neural networks called Siamese Networks [8]. Qian et al. [16] used them to verify the result of authorship attribution on C50 [3] and Guternberg [15] datasets. Their model consists of two gated recurrent units

978-3-903176-24-9

which take word vectors as inputs that are formed using 50-dimensional Glove [17] embeddings. Subsequently, Parikh et al. [18] implemented an architecture where they used skip-gram embeddings of words, character tri-grams and combinations of these two approaches to train their encoder and learn the embeddings from the data. Later these embeddings are used for classification using SVM, KNN, their cohort algorithm which is the binary classification of one author against each of the other authors, and a dense neural network. Their word based ensemble model produced the best results [19]. More recently, Boenninghoff et al. [20] constructed a hierarchical LSTM based Siamese Networks to compare text from two authors. They used 300-dimensional Glove [17] embeddings as the input vector. They evaluated their results on the dataset used by Halvani et al. [21] and showed that they outperformed the previous work by 15%.

Most of the earlier works in this area focus on longer texts and feature extraction techniques like character and word embeddings. These, as stated earlier, do not work effectively for short text messages. To this end, we propose a sub-word embedding technique which alongside Siamese Networks works better on short text data. We also study other feature extraction techniques and evaluate them in terms of their performance as the number of tweets and users change.

III. Methodology

Here we introduce the dataset [4], the feature extraction techniques and the neural network architectures used to implement the proposed system.

A. Dataset

The original dataset consists of over 7000 users with each user having 1000 tweets. In this paper, we randomly selected users from this dataset. For each tweet in the dataset, we replaced the word *@user* with *user*. We also replaced all the numbers and the URLs with *num* and *URL*, respectively. We considered punctuation as valid tokens and put space between them and any other words. Additionally, we removed multiple spaces and any other special characters. Lastly, we marked the start of the message with the word "BEGIN" and the end of the message with the word "END". The features are represented using word n-grams, character n-grams, flexible patterns and word embeddings. All these features are combined to form the input to the model.

B. Word and Character N-grams

N-grams are consecutive sequences of words and characters in text which can potentially capture repeating phrases shown to be useful for authorship attribution. We consider word n-grams of length $2 \leqslant n \leqslant 5$ and character n-grams of length $3 \leqslant n \leqslant 4$. However, we have not included the n-gram sequences that are in more than 95% of the messages since they do not provide uniqueness.

The n-grams are vectorized using term-frequency and inverse document frequency (tf-idf) [22] with sub-linear scaling. The tf-idf score is used to assign a weight to a word in the tweet based on the number of times it appears in a tweet and the number of tweets in which it appears. Eq. 1 shows tf-idf with sublinear scaling.

$$tf - idf_{t,d} = (1 + \log tf_{t,d}) \cdot \log \frac{N}{df_t} \tag{1}$$

where $tf_{t,d}$ is the number of times term t appears in a tweet d, N is the total number of tweets, and df_t is the number of tweets in which the term t appears. The maximum number of features considered after vectorizing with tf-idf are 40,000 for both character and word n-grams.

C. Flexible Patterns

Flexible patterns tag high frequency words and common words separately using a threshold which depends on the vocabulary size. Fig 1 describes the algorithm for generating the flexible pattern of a tweet. First, the total number of tokens for a user (author) is calculated along with frequency of each token. Next, we replaced words whose count is less than the threshold with a keyword "CW". These words are rare in occurrences and are called Content Words (CW). The threshold is selected as square root of log of total number of words in the training dataset vocabulary. For example, the sentence "I have a bat" will converted to "I have a CW" if bat is the content word. For vectorization, TF-IDF is used.

Data: T = given tweet
C = corpus
V = vocabulary of corpus
Result: Find the flexible pattern for the tweet
$freq \leftarrow \{\}$
for $word\ in\ V$ **do**
  $freq[word] \leftarrow\ count\ of\ word\ in\ C$
end
$thresh \leftarrow 2\log_{10} V$
$flex\_pattern \leftarrow []$
for $word\ in\ tweet$ **do**
  if $freq[word] \leqslant thresh$ **then**
    $flex\_pattern \leftarrow\ flex\_pattern\ + "CW"$
  else
    $flex\_pattern \leftarrow\ flex\_pattern\ + word$
end
$flex\_pattern \leftarrow flexible\ pattern\ for\ tweet\ T$

Algorithm 1: Pseudo-code for the flexible pattern

D. FastText

We use fastText [23] word embeddings in our models. FastText models can be trained to create a word embedding by making use of character level representations. These models learn a vector representation for a word by learning a representation for each of its character n-grams. Therefore, the overall word embedding is a weighted sum of the embeddings of these character n-grams. For example, for n=3, the vector for the word $hello$ would be represented by a sum of trigrams: $< he, hel, ell, llo, lo >$ where $<$ and $>$ denote the beginning and end of a word. The word embedding of word V is given by Eq. 2.

$$V = \sum_{n=1}^{N} w(n) \times v(n) \tag{2}$$

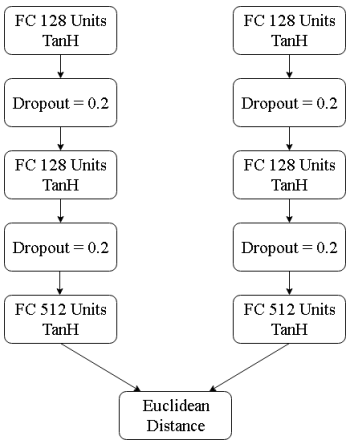

Fig. 1: Siamese network model for compromised tweet learning. During the training process, pairs with tweets of the same author are trained with 1 as supervisory signal and pairs of different authors are trained 0 as supervisory signal. The weights for both branches are shared to learn a common representation of tweets.

where $v(n)$ is the embedding vector for n^{th} n-gram of a word and $w(n)$ is the weight for the embedding. We use the skip-gram architecture when training the fastText embedding. The skip-gram contains a single hidden layer. The weights of this network are learnt by giving a word in the middle of a sentence as input. Then, use the surrounding words in a specific window size as supervisory signals. The word embedding from the skip-gram model is weighted using tf-idf of the word.

E. Models

1) Siamese Networks: Siamese Networks were used for one-shot image classification by Koch et al. [8]. They used Siamese Networks which consist of two sub-CNNs with shared weights. Pairs of images from the same class and also from different classes were created in equal proportion for a single batch of training. The image pairs are presented to the CNNs in the Siamese Network. The structure of the CNN was a series of convolution and pooling layers. The feature representations from the layers are flattened to a one-dimensional vector. The flattened vector represents the projection of the image onto a continuous vector space. The distance between the two vectors is calculated using the Euclidean distance [24]. This distance is fed into a fully connected layer and then finally, optimized using the cross-entropy loss function. The results show the network correctly differentiating pairs that are from the same class and those that are from different classes.

In our approach, we aim to determine if a given tweet is written by a given user. To this end, a representation of the user's style of writing is created by passing all tweets of a user into a MultiLayer Perceptron (MLP). Subsequently, the tweet whose author is to be determined is passed on to another MLP. The euclidean distance between the representation from both of these MLPs is calculated. During training, this signal is 1, if it is from the same user; and is 0, if it is from different users. During testing, we observe the euclidean distance and categorize it as the same user, if the value is greater than 0.5; and as a different user, if the value is less than 0.5. The contrastive loss [25] is defined in Eq. 3.

$$Loss = \begin{cases} \frac{1}{2}{||f_i - f_j||_2}^2 & \text{if } y_{ij} = 1 \\ \frac{1}{2} * max(0, m - {||f_i - f_j||_2}^2) & if y_{ij} = 0 \end{cases} \quad (3)$$

Here, f_i and f_j are representations of tweets i and j. y_{ij} is a boolean variable which is 1 when both tweets are from the same user, and 0 when the tweets are from different users. The contrastive loss requires the distance between embedded representations of tweets to be larger than a margin, m.

During the training process, we randomly select a user and 50 tweets of that user from the dataset. Each tweet is preprocessed by vectorizing with word n-grams, character n-grams and flexible pattern models. The preprocessed tweets are concatenated and passed on to the user network. Subsequently, a random tweet is selected and labeled with 1 (same user) and 0 (different users) signals.

IV. Experiments and Results

The proposed system is trained for 200 epochs with a patience number of 50. Based on empirical tests, our model converges well before 200 epochs, so we choose this as the upper limit. For feature extraction, we combine the word, character and flexible n-grams in a single stack and consider this as one input. In comparison to this, we also consider just the tf-idf weighted word embedding feature set as the input. The word embeddings have a maximum of 100 dimensions as we are taking the mean of every word in the dataset.

The training process is divided into two phases. In phase 1, the total number of users is 50 and the number of tweets ranges from 10 to 100. This is done to compare the aforementioned extraction techniques. In phase 2, the total number of tweets is 100 and the number of users ranges from 10 to 100. This is done to test whether the performance of the model is impervious to the number of users. We randomly split 10% of the dataset for testing and the remaining 90% is used for training. While training, all the tweets from a particular user have label 1, if they are combined with the tweets of the same user and label 0 if they are combined with a randomly selected tweet from any other user. We keep an equal number of positive and negative samples to keep the training set balanced. Although, after every epoch, we include more random samples to make our network robust [8].

Fig 1 shows the architecture of the Siamese Network containing two branches of a MLP. The MLP has four fully connected layers, where three of them have 128 units with Tanh activation function, while the last layer having 512 units. We have used dropout layers with rate 0.1 for regularization. The last layer is a lambda layer which uses output from both branches as input and calculates the euclidean distance between the two representations. We pass the combined representation of users into the left network and the user tweet that is tested for compromised detection into the right. We use contrastive loss with a margin of 1. We use the ADAM optimizer [26] with a learning rate of 0.001.

model	word, char and flex		fastText embedding		
#tweets	mean	std dev	mean	std dev	p-value
10	0.57	0.03	0.60	0.04	0.14
20	0.65	0.02	0.70	0.05	0.02
30	0.66	0.02	0.69	0.05	0.06
50	0.66	0.01	0.71	0.03	0.002
100	0.71	0.01	0.73	0.04	0.12

TABLE I: Accuracy for 50 users as the number of tweets changes.

Fig 2 shows the test accuracy as the number of tweets changes for 50 users using Siamese Networks. We employ two types of features as input to the Siamese Network. First, we vectorize the tweets using word n-grams, character n-grams and flexible patterns. Secondly, we use the fastText word embedding tweets with a dimension of 100. We trained both models 10 times and calculated the mean test accuracy as shown in Fig 2. The mean test accuracy increases as the number of tweets increases.

A student t-test between test accuracies for word, char and flexible pattern models with 10 tweets and 20 tweets gives a p-value of $2.5 \times e^{-5}$. This p-value is less than 0.05, showing that the we can reject the null hypothesis and data have been drawn from different distributions. Therefore, the test accuracy for 20 tweets is greater than 10 tweets. Although, there is no major difference between the test accuracies of 20, 30 and 50 tweets. However, when we increase the number of tweets to 100, we observe an increase in the mean test accuracy to 71% (1% std). A student t-test between test accuracies of 50 tweets and 100 tweets gives p-value of $2.8 \times e^{-10}$, rejecting the null-hypothesis. This shows that there is a difference between the test accuracies for 50 and 100 tweets.

Table I shows a comparison between test accuracies for word, char and flexible pattern model and fastText word embedding model. The test accuracies are similar for 10,30 and 100 tweets with p-value greater than 0.05. However, fastText word embedding model performs better with 20 and 50 tweets. The fastText model input has a lower dimension than the word, char and flexible pattern models.

Fig 3 shows the test accuracy as the number of users (each trained with 100 tweets) changes using Siamese Networks. We use the fastText word embedding of tweets with a dimension of 100 as input to the Siamese Networks. We ran the training process 10 times and calculated the mean test accuracy as shown in Table II. The increase in the number of users does not impact the performance of the Siamese Networks. The mean test accuracy is similar for 10, 20 and 30 user. The test accuracies of 50 and 100 users exceed the test accuracy of 30 users. A student t-test between test accuracies with 50 users and 30 users gives a p-value of 0.001. This p-value is less than 0.05, showing that we can reject the null-hypothesis and data have been drawn from different distributions.

V. Conclusion and Future Works

We observe that compromised user detection using Siamese Networks performs well as the number of tweets increases. The mean test accuracy improves as the number of tweets increases, Fig. 2. The test accuracy performance does not degrade as the number of users increases as shown in Fig. 3. We also perform a comparison between the fastText word embedding model and the model that used word, char and flex n-grams as the feature representations, Table I. The test accuracies are similar across both models except for the fastText model, which has a lower dimensional input resulting in a lower computational cost.

For future work, we plan to use auto-encoder models to create another latent representation of the tweets before passing them as input to a Siamese Neural Network.

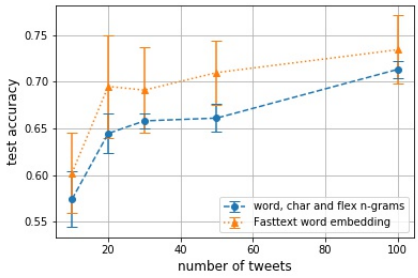

Fig. 2: Test accuracy vs the number of tweets.

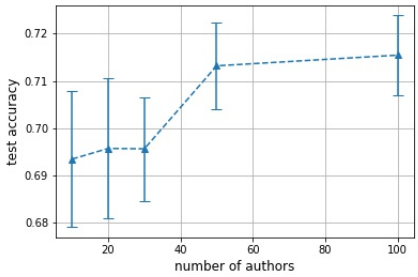

Fig. 3: Test accuracy vs the number of users (authors)

Acknowledgments

This research is supported by the Canadian Safety and Security Program (CSSP) E-Security grant. The CSSP is led by the Defense Research and Development Canada, Centre for Security Science (CSS) on behalf of the Government of Canada and its partners across all levels of government, response and emergency management organizations, nongovernmental agencies, industry and academia.

References

[1] Mike Kestemont, Michael Tschuggnall, Efstathios Stamatatos, Walter Daelemans, Günther Specht, Benno

model	fastText embedding	
#authors	mean	std dev
10	0.69	0.01
20	0.69	0.01
30	0.69	0.01
50	0.71	0.01
100	0.71	0.01

TABLE II: Accuracy for different users, each trained with 100 tweets.

Stein, and Martin Potthast. Overview of the author identification task at pan-2018: cross-domain authorship attribution and style change detection. In *Working Notes Papers of the CLEF 2018 Evaluation Labs. Avignon, France, September 10-14, 2018/Cappellato, Linda [edit.]; et al.*, pages 1–25, 2018.

[2] Efstathios Stamatatos. Authorship attribution using text distortion. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1138–1149, 2017.

[3] T.G. Rose, M. Stevenson, and M. Whitehead. The reuters corpus volume 1-from yesterdays news to tomorrows language resources. *Proceedings of the Third International Conference on Language Resources and Evaluation*, pages 29–31, 2002. URL http://about.reuters.com/researchandstandards/corpus/LREC_camera_ready.pdf.

[4] Roy Schwartz, Oren Tsur, Ari Rappoport, and Moshe Koppel. Authorship attribution of micro-messages. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1880–1891, 2013.

[5] Prasha Shrestha, Sebastian Sierra, Fabio Gonzalez, Manuel Montes, Paolo Rosso, and Thamar Solorio. Convolutional neural networks for authorship attribution of short texts. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, volume 2, pages 669–674, 2017.

[6] Tien D Phan and Nur Zincir-Heywood. User identification via neural network based language models. *International Journal of Network Management*, page https://doi.org/10.1002/nem.2049, 2018.

[7] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[8] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML Deep Learning Workshop*, volume 2, 2015.

[9] Shota Horiguchi, Daiki Ikami, and Kiyoharu Aizawa. Significance of softmax-based features in comparison to distance metric learning-based features. *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[10] Robert Layton, Paul Watters, and Richard Dazeley. Authorship attribution for twitter in 140 characters or less. In *2010 Second Cybercrime and Trustworthy Computing Workshop*, pages 1–8. IEEE, 2010.

[11] Georgia Frantzeskou, Efstathios Stamatatos, Stefanos Gritzalis, Carole E Chaski, and Blake Stephen Howald. Identifying authorship by byte-level n-grams: The source code author profile (scap) method. *International Journal of Digital Evidence*, 6(1):1–18, 2007.

[12] Marti A. Hearst, Susan T Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. Support vector machines. *IEEE Intelligent Systems and their applications*, 13(4): 18–28, 1998.

[13] Dmitry Davidov and Ari Rappoport. Efficient unsupervised discovery of word categories using symmetric patterns and high frequency words. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 297–304. Association for Computational Linguistics, 2006.

[14] Dylan Rhodes. Author attribution with cnns. *Avaiable online: https://www. semanticscholar. org/paper/Author-Attribution-with-Cnn-s-Rhodes/0a904f9d6b47dfc574f681f4d3b41bd840871b6f/pdf (accessed on 22 August 2016)*, 2015.

[15] Michael Hart. Project gutenberg. https://www.gutenberg.org/wiki/Main_Page, 1971. (Accessed on 06/02/2019).

[16] Chen Qian, Tianchang He, and Rao Zhang. Deep learning based authorship identification. 2017.

[17] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL http://www.aclweb.org/anthology/D14-1162.

[18] Kieran Sagar Parikh, Vinodini Venkataram, and Jugal Kalita. Towards a universal document encoder for authorship attribution. 2018.

[19] John Houvardas and Efstathios Stamatatos. N-gram feature selection for authorship identification. In *International conference on artificial intelligence: Methodology, systems, and applications*, pages 77–86. Springer, 2006.

[20] Benedikt Boenninghoff, Robert M Nickel, Steffen Zeiler, and Dorothea Kolossa. Similarity learning for authorship verification in social media. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2457–2461. IEEE, 2019.

[21] Oren Halvani, Christian Winter, and Anika Pflug. Authorship verification for different languages, genres and topics. *Digital Investigation*, 16:S33–S43, 2016.

[22] Gerard Salton and Michael J McGill. Introduction to modern information retrieval. 1986.

[23] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[24] Per-Erik Danielsson. Euclidean distance mapping. *Computer Graphics and image processing*, 14(3):227–248, 1980.

[25] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *null*, pages 1735–1742. IEEE, 2006.

[26] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.