MADAFE: Malicious Account Detection on Twitter with Automated Feature Extraction

Hao Yue
Department of Computer Science
San Francisco State University
San Francisco, USA
haoyue@sfsu.edu

Lijie Zhou
Department of Computer Science
San Francisco State University
San Francisco, USA
lijie@mail.sfsu.edu

Kaiping Xue
School of Cybersecurity
University of Science and Technology of China
Hefei, China
kpxue@ustc.edu.cn

Hongning Li
Telecommunication Engineering Department
Xidian University
Xi'an, China
hnli@xidian.edu.cn

***Abstract*—Nowadays, more and more massive cyber-attacks have been launched over social networks. Using compromised or fake accounts, criminals can exploit the inherent trust between connected users to effectively spread malicious content and perform scams against users. Detecting those malicious accounts on social networks like Twitter has received increasing attention from government, industry, and academia. Traditional methods on malicious account detection often leverage features that are created and selected based on domain knowledge of user data, which is inefficient, time-consuming, and possibly biased due to different understanding and observations of the data. In this paper, we propose a new framework, called MADAFE, for accurate and efficient malicious account detection on social networks like Twitter. To overcome the limitation of existing work on manual feature extraction, MADAFE utilizes an autoencoder to automate feature extraction and selection from unlabeled user data. A softmax regression model is also established and trained with the extracted features for classification of benign and malicious accounts. We test MADAFE on different datasets, and extensive simulation results show that MADAFE is effective in detecting malicious accounts, which outperforms state-of-the-art detection methods.**

***Index Terms*—malicious account detection, social networks, feature extraction, autoencoder**

I. Introduction

Online social networks have become one of the most popular communication tools for Internet users to interact with each other, which allows them to broadcast breaking news, share ideas or moments, comment on friends' posts, etc. As of June 2019, there are on average 1.59 billion daily active users on Facebook and more than 130 million daily active users on Twitter. The convenience and popularity of online social networks, however, also attract cybercriminals, which launch massive cyber-attacks through creation and dissemination of faked or harmful messages to users for malicious purposes [2]. It was estimated that 7% of Twitter accounts were faked in 2013. This number increased to 15% in 2017, which means nearly 48 million users on Twitter were fake accounts.

Detecting malicious accounts on social networks has received increasing attention from government, industry, and academia. Most of recent solutions [9]–[12] address this problem with machine learning. These solutions collect data from user accounts and extract statistical features, such as the number of followers, the number of tweet messages, the number of times a tweet message has been retweeted, etc., to characterize their behaviors and differentiate malicious accounts from benign accounts. Then, a machine learning model is trained with those extracted features to detect unknown malicious accounts on social networks. There are several drawbacks of the existing solutions. First, the features for malicious account detection are extracted and selected based on the domain knowledge of researchers on the user account data that are collected, which might easily induce bias to the extracted features due to different understanding and observations of the data. There is no universal standard on feature extraction for malicious account detection, and hence it is common that a set of features work well on one dataset, but fail on another one. Second, to achieve high detection accuracy, current solutions often need vast amount of user account data with labels to train machine learning classifiers. However, it is extremely time-consuming and labor-intensive to collect and label large number of user accounts, since the labeling cannot be automated as yet, which significantly restricts the implementation of machine learning based solutions in practice.

In this paper, we propose a new framework, called MADAFE, to accurately and efficiently detect malicious accounts on social networks like Twitter. To overcome the limitations of manual feature extraction in existing work, MADAFE utilizes an autoencoder to automate feature extraction and selection with unlabeled user data. A softmax regression model is also established and trained with the extracted features for classification of benign and malicious accounts. The main contributions of this work is summarized as follows:

This work was supported in part by the Key Program of NSFC-Tongyong Union Foundation under Grant U1636209, the National Natural Science Foundation of China under Grant 61902292.

- We develop a new framework for malicious account detection on social networks like Twitter.
- We utilize an autoencoder to automatically extract and select salient features from unlabeled user account data, which eliminates the need of labeling vast amount of user account data for training machine learning based classifiers and reduces the cost of manual feature extraction and selection in traditional detection methods.
- Extensive evaluation results demonstrate that the proposed framework is effective in detecting malicious accounts in terms of accuracy and cost, and it also outperforms other existing machine learning based solutions.

The rest of this paper is organized as follows. The related work is reviewed in Section II. In Section III, we elaborate the design of the new framework for malicious account detection. We conduct experiments and evaluate the performance of the proposed scheme in Section IV. Finally, we draw the concluding remarks in Section V.

II. Related Work

A. Malicious Message Detection

Malicious message detection, also known as spam detection, screens individual tweet messages for harmful or spam content or URLs, which has been extensively studied in the literature. Many earlier detection methods [3]–[5] focus on URLs and shortened URLs in tweet messages. For instance, Thomas et al. [4] collect and extract features such as initial URL, final URL, frame URLs, and source URLs from tweet messages via web browser to differentiate malicious and benign messages. In [5], Lee and Kim derive features from URLs and the correlation of URL redirected chains and train a statistical classifier to detect malicious messages in the Twitter stream in real time. Some recent work also take social behavior of user accounts into consideration [6]. Gao et al. extract features on the social degree of the sender of a tweet message and his interaction history for online filtering of malicious messages. In [7], Wu et al. build a word2vec model to learn text-based features from each tweet message. Readers are referred to the survey [8] for more related work on malicious message detection on social networks.

B. Malicious Account Detection

Different from malicious message detection, malicious account detection examines individual accounts for harmful or abusive behavior, such as spreading spam, scam, or phishing tweets. In [9], Benevenuto et al. identified a number of characteristics related to tweet content and user social behavior, and used them as features to train supervised learning models to detect malicious accounts. Lee et al. [13] deployed honeypots on social networks to harvest malicious accounts, extracted behavior-based features from accounts as well as content-based features from messages, and established machine learning based classifier to identify spammer. Yang et al. [16] introduced new features including graph-based features, neighbor-based features, timing-based features, and automation-based features to machine learning based detection methods. Wang considered both graph-based features and content-based features for the classification of malicious users and benign users [14]. Egele et al. [10] built behavioral profiles for users based on features extracted from their historical activities to characterize their normal behavior, and screened new messages for sudden changes in behavior to detect compromised accounts. Stringhini et al. [12] observed the activities of collected malicious accounts on social networks and created statistical features, including the number of friends, the number of tweets, friend choice, and URL ratio, to differentiate malicious and legitimate users. In [15], Hu et al. introduced a unified mathematical framework on malicious account detection that considers both content information and network information, and applied online learning to incrementally update the model for efficient large-scale online detection.

III. Malicious Account Detection with Automated Feature Extraction

In this section, we describe the design of a new framework, called MADAFE, for malicious account detection and elaborate the methods used in MADAFE to achieve automated feature extraction and accurate classification of benign and malicious accounts.

A. Overview

The overview of the new framework MADAFE is shown in Fig. 1. We assume that two datasets are collected: one is an unlabeled dataset, which is denoted as $\mathcal{X} = \{X_1, X_2, ..., X_m\}$, where each sample X_i contains the data of a user account; the other one is a labeled dataset, which is denoted as $\mathcal{T} = \{(Y_1, z_1), (Y_2, z_2), ..., (Y_n, z_n)\}$, where Y_j contains the data of a user account and $z_j \in \{0, 1\}$ is the label of that user. Here, $z_j = 1$ indicates that the user is a spammer, and $z_j = 0$ indicates that the user is benign. As shown in Fig. 1, the framework consists of offline training and online classification. In offline training, we create an autoencoder and train it with the unlabeled dataset $\mathcal{X}$ to enable automated feature extraction from user data. Then, we extract features from the labeled dataset $\mathcal{T}$ with the trained autoencoder and use the features to train a machine learning model for malicious account detection. In online classification, when there is a user account to be examined, we feed his data into the autoencoder and the machine learning classifier, which will return a result on whether the user is malicious or not.

B. Feature Extraction with Autoencoder

As we discussed in Sec. I, most of existing machine learning based solutions on malicious account detection create and select features based on domain knowledge or observations of user data, which is time-consuming and probably biased. Different from those methods with manual feature extraction, in this subsection, we utilize an autoencoder to automate feature extraction for the classification of benign and malicious users. An autoencoder is a neural network that can learn efficient compressed knowledge representation of the original data in an unsupervised manner. Fig. 2 shows the architecture

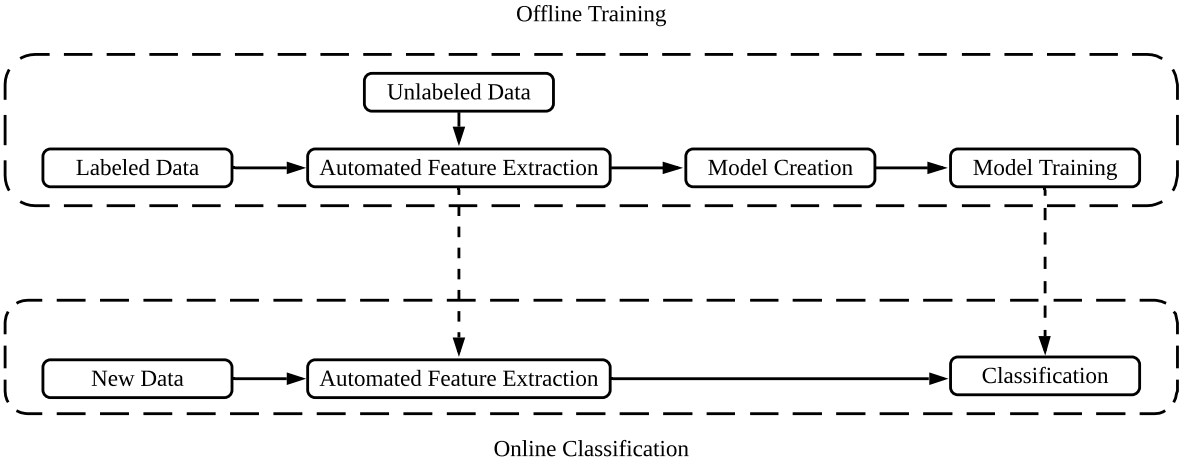

Fig. 1. Framework overview

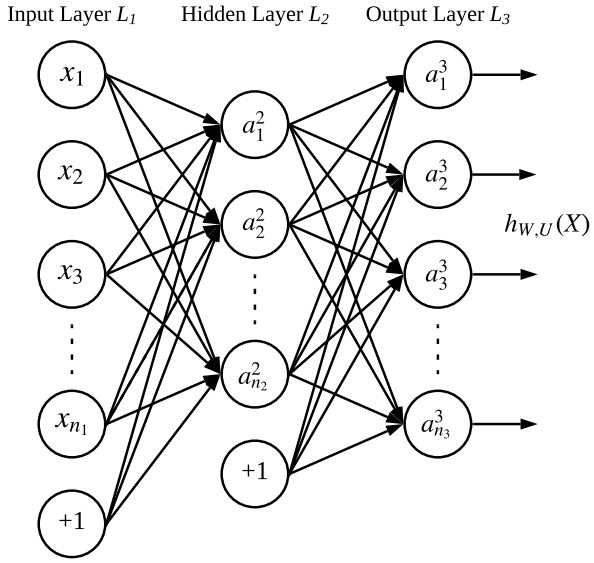

Fig. 2. Autoencoder

of the autoencoder. It consists of three layers: the input layer, the hidden layer and the output layer, which are denoted as $\mathcal{L}_1$, $\mathcal{L}_2$, and $\mathcal{L}_3$, respectively. Each of them is comprised of a number of computational units, called neurons. We denote the number of neurons in the input layer, the hidden layer, and the output layer as n_1, n_2, and n_3, respectively. There is also a unit labeled "+1" in the input layer and the hidden layer, which is called bias unit and takes the value of +1. As illustrated in Fig. 2, each neuron in the input layer takes a value in the input sample. Each neuron in the hidden layer is connected with all the neurons and the bias unit in the input layer, which takes their values as inputs and computes an output as follows:

$$a_j^2 = f(\sum_{i=1}^{n_1} w_{ij}^1 x_i + u_j^1), \tag{1}$$

where a_j^2 denotes the output of the jth neuron in the hidden layer, w_{ij}^1 denotes the weight associated with the connection between the ith neuron in the input layer and the jth neuron in the hidden layer, and u_j^1 denotes the weight associated with the connection between the bias unit in the input layer and the ith neuron in the hidden layer. Here, $f : \mathbb{R} \to \mathbb{R}$ is the activation function. There are different kinds of activation functions we can choose. In this work, we use the sigmoid function as the activation function, which is defined as follows:

$$f(x) = \frac{1}{1 + \exp(-x)}.$$

Similarly, each neuron in the output layer connects with all the neurons and the bias unit in the hidden layer, and takes their values as inputs to calculate an output. Let a_k^3 denote the output of the kth neuron in the output layer, w_{jk}^2 denote the weight of the connection between the jth neuron in the hidden layer and the kth neuron in the output layer, and u_k^2 denote the weight of the connection between the bias unit in the hidden layer and the kth neuron in the output layer. Then, the output of the kth neuron in the output layer can be written as

$$a_k^3 = f(\sum_{j=1}^{n_2} w_{jk}^2 a_j^2 + u_k^2). \tag{2}$$

Substituting a_j^2 in (2) with (1), the output of the kth neuron in the output layer of the autoencoder can be rewritten as

$$a_k^3 = f\big[\sum_{j=1}^{n_2} w_{jk}^2 \cdot f(\sum_{i=1}^{n_1} w_{ij}^1 x_i + u_j^1) + u_k^2\big]. \tag{3}$$

Let $\mathcal{W}$ denote the set of all the weights associated with the connections between neurons in the autoencoder. Let $\mathcal{U}$ denote the set of weights associated with the connections from bias units to neurons in the autoencoder. Given an input sample $X = (x_1, x_2, ..., x_{n_1})$ to the autoencoder, according

to (3), it can be observed that the final output of the autoencoder is a function of $\mathcal{W}$ and $\mathcal{U}$. We define $h_{\mathcal{W},\mathcal{U}}(X)$ as the output vector of the output layer of the autoencoder, i.e., $h_{\mathcal{W},\mathcal{U}}(X) = (a_1^3(X), a_2^3(X), ..., a_{n_3}^3(X))$, where $a_k^3(X)$ $(1 \leq k \leq n_3)$ denotes the value of the kth neuron in the output layer when the input to the autoencoder is X. The autoencoder aims to find the optimal parameters $\mathcal{W}$ and $\mathcal{U}$ such that the output of the autoencoder is approximately equal to the value of the input sample, i.e., $h_{\mathcal{W},\mathcal{U}}(X) \approx X$. When this condition is satisfied and the number of neurons in the hidden layer is small, the autoencoder is able to learn a compressed representation of input data at the hidden layer, which can then be used as features for machine learning based classification.

The optimal values for the weight parameters $\mathcal{W}$ and $\mathcal{B}$ of the autoencoder can be calculated using the back-propagation algorithm [18]. Considering an unlabeled training set $\mathcal{X} = \{X_1, X_2, ..., X_m\}$ of m training samples, we define a cost function as follows:

$$\begin{aligned} J(\mathcal{W},\mathcal{U}) = &\left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\left\|h_{\mathcal{W},\mathcal{U}}(X_i) - X_i\right\|^2\right)\right] \\ &+ \frac{\lambda}{2}\Big[\sum_{i=1}^{n_1}\sum_{j=1}^{n_2}\left(w_{ij}^1\right)^2 + \sum_{j=1}^{n_2}\sum_{k=1}^{n_3}\left(w_{jk}^2\right)^2\Big]. \end{aligned} \tag{4}$$

Here, the first term in the cost function $J(\mathcal{W},\mathcal{U})$ calculates the average sum of squared errors between the output of the autoencoder and the input samples. and the second term is the regularization term to avoid overfitting. λ is a parameter that determines the relative weights of these two terms in the cost function.

In an autoencoder, the number of neurons in the hidden layer must be smaller than that in the input layer to ensure that the hidden layer can learn compressed representation (i.e., features) from the input data. One method to achieve this is to test all the values that are smaller than the number of neurons in the input layer as the size of the hidden layer, and find the one that brings the best performance. In this work, we take an alternative method which enables the hidden layer to learn latent features with a small number of activated neurons, rather than reducing the total number of neurons in the hidden layer. Here, a neuron is called active when its output value is close to 1, and it is called inactive when its output value is close to 0. Given the training set $\mathcal{X} = \{X_1, X_2, ..., X_m\}$, the average output value of the jth neuron in the hidden layer of the autoencoder over all the training samples in $\mathcal{X}$, which is denoted as $\overline{\rho}_j$, can be calculated as

$$\overline{\rho}_j = \frac{1}{m}\sum_{i=1}^{m}\left[a_j^2(X_i)\right].$$

We introduce a sparsity parameter ρ, which is a small value close to 0, and enforce the sparsity constraint $\overline{\rho}_j = \rho$ on all the neurons in the hidden layer to ensure that the average activation of each neuron in the hidden layer is close to 0. To achieve this, we add an extra term to the cost function of the autoencoder $J(\mathcal{W},\mathcal{U})$ to penalize the derivation of the average output of a neuron $\overline{\rho}_j$ from the sparsity parameter ρ. In this work, we use Kullback-Leibler ($\mathbf{KL}$) divergence to measure the difference between $\overline{\rho}_j$ and ρ, which is defined as follows:

$$\mathbf{KL}(\rho||\overline{\rho}_j) = \rho\log\frac{\rho}{\overline{\rho}_j} + (1-\rho)\log\frac{1-\rho}{1-\overline{\rho}_j}.$$

It is observed that the divergence function $\mathbf{KL}(\rho||\overline{\rho}_j)$ will increase as the difference between $\overline{\rho}_j$ and ρ increases. Therefore, by minimizing the divergence function $\mathbf{KL}(\rho||\overline{\rho}_j)$, we can enforce $\overline{\rho}_j$ to be close to 0, and make sure most of neurons in the hidden layer are inactive.

After adding the above $\mathbf{KL}$ divergence term into the cost function $J(\mathcal{W},\mathcal{U})$, we obtain the cost function for the sparse autoencoder, which is shown as follows

$$\begin{aligned} \widehat{J}(\mathcal{W},\mathcal{B}) = &\left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\left\|h_{\mathcal{W},\mathcal{U}}(X_i) - X_i\right\|^2\right)\right] \\ &+ \frac{\lambda}{2}\Big[\sum_{i=1}^{n_1}\sum_{j=1}^{n_2}\left(w_{ij}^1\right)^2 + \sum_{j=1}^{n_2}\sum_{k=1}^{n_3}\left(w_{jk}^2\right)^2\Big] \\ &+ \beta\sum_{j=1}^{n_2}\mathbf{KL}(\rho||\overline{\rho}_j). \end{aligned} \tag{5}$$

where β is the weight of the Kullback-Leibler divergence term in the cost function. Based on this cost function, the parameters $\mathcal{W}$ and $\mathcal{B}$ can be computed with the iterative gradient descent algorithm. After we determine $\mathcal{W}$ and $\mathcal{B}$ for the autoencoder, the output vector of the hidden layer, denoted as $(a_1^2, a_2^2, ..., a_{n2}^2)$, contains the features of user data, which can be used to train machine learning models for classification.

C. Softmax Regression based Classification

Next, we create a softmax regression model and train it as a classifier with the features extracted from the above autoencoder to differentiate malicious and benign accounts. Softmax regression [17] is a generalization of logistic regression model, which is often used together with an autoencoder to solve classification problems. Considering a sample X and a set of classes, the softmax regression model aims to estimate the probabilities that X falls into each class. Since there are two classes in spammer detection: malicious and benign, the output of the softmax regression model in our framework, which is denoted as $h_{\Theta_0,\Theta_1}(X)$, can be written as

$$\begin{aligned} h_{\Theta_0,\Theta_1}(X) &= \begin{bmatrix} \text{Prob}(z=0|X;\Theta_0,\Theta_1) \\ \text{Prob}(z=1|X;\Theta_0,\Theta_1) \end{bmatrix} \\ &= \begin{bmatrix} \frac{\exp\left(\Theta_0^\top X\right)}{\exp\left(\Theta_0^\top X\right)+\exp\left(\Theta_1^\top X\right)} \\ \frac{\exp\left(\Theta_1^\top X\right)}{\exp\left(\Theta_0^\top X\right)+\exp\left(\Theta_1^\top X\right)} \end{bmatrix}. \end{aligned}$$

where $z = 0$ indicates that the data sample X comes from a malicious user, and $z = 0$ indicates that the user is benign. Θ_0 and Θ_1 are two vectors of weight parameters for the softmax regression model, and $\top$ denotes the transpose of a matrix.

As we described in Sec. III-A, we use the labeled dataset $\mathcal{T} = \{(Y_1, z_1), (Y_2, z_2), ..., (Y_n, z_n)\}$ to train the above softmax regression model. For each labeled sample (Y_i, z_i) $1 \leq i \leq n$, we feed it to the trained autoencoder and obtain a set of features, which is denoted as $A^2(Y_i) = (a_1^2(Y_i), a_2^2(Y_i), ..., a_{n_2}^2(Y_i))$. Then, we train the softmax regression model with the features $A^2(Y_i)$ and the labels z_i to find optimal parameters Θ_0 and Θ_1 such that the output of the softmax regression model $h_{\Theta_0,\Theta_1}(Y_i)$ matches the label z_i. We define a cost function to measure the average error between the actual labels and the prediction from the softmax regression model over the labeled dataset $\mathcal{T}$, which is shown as follows:

$$J(\Theta_0, \Theta_1) = -\frac{1}{n}\Bigg[\sum_{i=1}^{n}\sum_{k=0}^{1} \mathbf{1}\{z_i = k\} \log \frac{\exp\left(\Theta_k^\top A^2(Y_i)\right)}{\exp\left(\Theta_0^\top A^2(Y_i)\right) + \exp\left(\Theta_1^\top A^2(Y_i)\right)}\Bigg] + \frac{\beta}{2}\sum_{k=0}^{1}||\Theta_k||,$$

where the first term measures the average error between the actual labels and the hypothesis, and the second term is the regularization term to avoid overfitting. β controls the weight of the regularization term in the cost function. $\mathbf{1}\{\cdot\}$ is an indicator function where $\mathbf{1}\{\text{false statement}\} = 0$ and $\mathbf{1}\{\text{true statement}\} = 1$. By minimizing the cost function with gradient descent algorithm, we can determine the weight parameters Θ_0 and Θ_1 for the softmax regression model and use it as a classifier to detect malicious accounts.

IV. Evaluation

A. Dataset

We use the dataset collected in [9] to evaluate the effectiveness and cost of MADAFE in malicious account detection. This dataset has been used in the literature as the benchmark dataset to test new methods for malicious account detection on social networks. There are 1065 labeled Twitter accounts in this dataset and each account is associated with 62 attributes. The attributes can be classified into two categories: content attributes and user behavior attributes. Content attributes refer to the properties of the tweets posted by users, such as the fraction of tweets with spam words, the fraction of tweets with URLs, *etc.* User behavior attributes are the properties that reflect user behavior patterns, such as the number of followers and followees, the fraction of tweets replied, *etc.* 355 out of the 1065 user accounts are labeled as malicious, and the other 710 user accounts are labeled as benign. In our experiments, we consider all 62 attributes as features, and divide the dataset into three subsets: one training set with 600 user accounts for training the autoencoder, one training set with 365 user accounts for training the softmax regression model, and the testing set with 100 user accounts.

B. Performance Metrics

We consider four metrics when evaluating the performance of MADAFE, which are Accuracy, Precision, Recall, and F1 score. The definition of those four metrics are introduced as follows:

- Accuracy: the ratio of the number of correctly classified instances to the total number of instances.
- Precision: the ratio of the number of correctly classified malicious instances to the total number of instances that are classified as malicious.
- Recall: the ratio of the number of malicious instances that are correctly classified to the total number of malicious instances.
- F1 score: the weighted average of Precision and Recall, which measures the balance between them for a machine learning based classification model.

C. Results

We first evaluate the performance of MADAFE with different number of neurons in the hidden lyaer of the autoencoder. The detection accuracy is shown in Fig 3. It can be observed that MADAFE can always achieve high detection accuracy when the number of hidden neurons increases from 15 to 55. The reason is that we use the sparsity parameter ρ to control the number of active neurons and ensure it is small, and therefore the performance of MADAFE is stable under different sizes of the hidden layer.

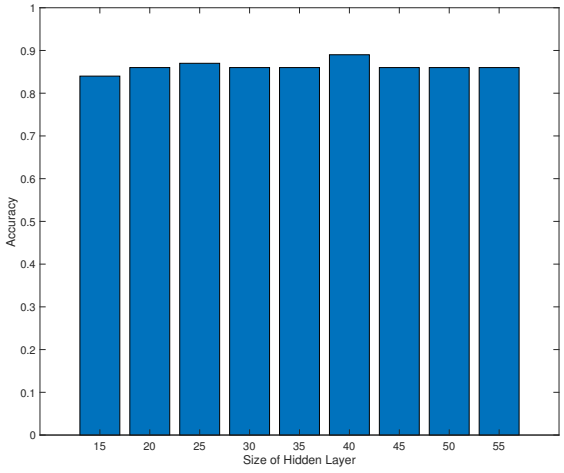

Fig. 3. Performance under different sizes of hidden layer

We also compare the performance of MADAFE with two classic machine learning models: Random Forest model and Naive Bayes model for malicious account detection on social networks. The results are shown in Fig. 4. It can be observed that MADAFE achieves better performance in terms of accuracy, precision, recall, and F1 score than those two machine learning models.

Next, we compare MADAFE with one state-of-the-art solution on malicious account detection that utilizes the Support

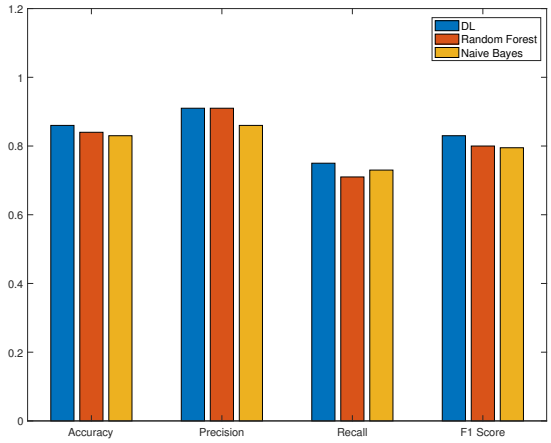

Fig. 4. Performance comparison with classic machine learning based solutions

Vector Machine (SVM) model to classify malicious and benign accounts [9]. The top 5, 10, and 20 features are identified with feature selection algorithm, which are used to train the SVM model. The evaluation results are illustrated in Fig. 5. As shown in Fig. 5, as the number of features increases, the accuracy, precision, recall, and F1 score measurements of the SVM-based solution also increases. Our framework MADAFE outperforms the SVM-based solution when 5, 10, and 20 features are considered.

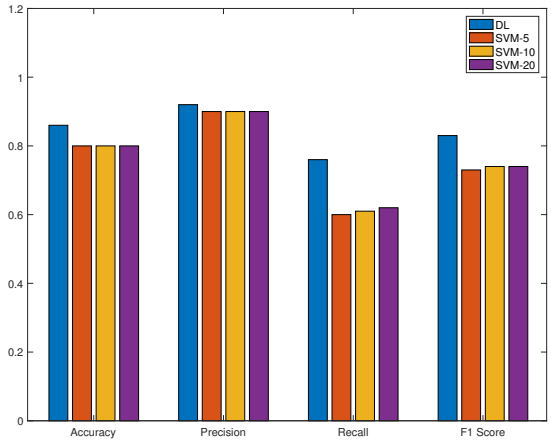

Fig. 5. Performance compasion with different number of features

V. Conclusion

In this paper, we present a new framework for achieving efficient and accurate malicious account detection on Twitter. In this framework, we establish and train an autoencoder to automatically extract and select salient features from unlabeled user account data, which eliminates the need of collecting vast amount of labeled account data for training machine learning based classifiers and reduces the cost of manual feature extraction and selection in traditional detection methods. Evaluation results demonstrate that the proposed framework is effective in detecting malicious accounts in terms of accuracy and cost, and it also outperforms other existing machine learning based solutions.

References

[1] "Twitter Blog, Update on Twitters Review of the 2016 U.S. Election," https://blog.twitter.com/official/en_us/topics/company/2018/2016-election-update.html

[2] C. Grier, K. Thomas, V. Paxson, and M. Zhang, "@ spam: the underground on 140 characters or less," In Processings of the 17th conference on Computer and Communications Security, Chicago, Illinois, 2010.

[3] Z. Chu, I. Widjaja, and H. Wang, "Detecting Social Spam Campaigns on Twitter," in *Proceedings of the International Conference on Applied Cryptography and Network Security (ACNS)*, 2012.

[4] K. Thomas, C. Grier, J. Ma, V. Paxson, and D. Song, "Design and Evaluation of a Real-Time URL Spam Filtering Service," in *Proceedings of the 32nd IEEE Symposium on Security & Privacy*, 2011.

[5] S. Lee and J. Kim, "WarningBird: Detecting Suspicious URLs in Twitter Stream," in *Proceedings of the 19th Annual Network & Distributed System Security Symposium (NDSS)*, 2012.

[6] H. Gao, Y. Chen, K. Lee, D. Palsetia, and A. Choudhary, "Towards Online Spam Filtering in Social Networks," in *Proceedings of the 19th Annual Network & Distributed System Security Symposium (NDSS)*, 2012.

[7] T. Wu, S. Liu, J. Zhang, and Y. Xiang, "Twitter Spam Detection based on Deep Learning," in *Proceedings of the Australasian Computer Science Week Multiconference (ACSW)*, 2017.

[8] T. Wu, S. We, Y. Xiang, and W. Zhou, "Twitter Spam Detection: Survey of New Approaches and Comparative Study," *Elsevier Computers & Security*, vol. 76, pp. 265-284, July 2018.

[9] F. Benevenuto, G. Magno, T. Rodrigues, and V. Almeida, "Detecting spammers on Twitter," Collaboration, Electronic messaging, Anti-Abuse and Spam Conference (CEAS), 2010.

[10] M. Egele, G. Stringhini, C. Kruegel, and G. Vigna, "COMPA: Detecting Compromised Accounts on Social Networks," In NDSS 2013.

[11] L. Liu, Y. Lu, Y. Luo, R. Zhang, L. Itti, and J. Lu, "Detecting "Smart" Spammers On Social Network: A Topic Model Approach presented," atXiv:1604.08504 (2016).

[12] G. Stringhini, C. Kruegel, and G. Vigna, "Detecting spammers on social networks," In Proceeding of the 26th Annual Computer Security Applications Conference, 2010.

[13] K. Lee, J. Caverlee, and S. Webb, "Uncovering Social Spammers: Social Honeypots + Machine Learning," In Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval (SIGIR'10), 2010.

[14] G. Stringhini, C. Kruegel, and G. Vigna, "Don't follow me: Spam detection in Twitter," In Proceeding of 2010 International Conference on Security and Cryptography (SECRYPT), 2010.

[15] X. Hu, and J. Tang, and H. Liu, "Online Social Spammer Detection," In Proceeding of 28th AAAI Conference on Artificial Intelligence (AAAI'14), 2014.

[16] C. Yang, and R. Harkreader, and G. Gu, "Empirical Evaluation and New Design for Fighting Evolving Twitter Spammers," IEEE Transactions on Information Forensics and Security, vol. 8, pp. 1280–1293, August 2013.

[17] D. Heckerman and C. Meek, "Models and selection criteria for regression and classification," in Proc. 13th Int. Conf. Uncertainty in Artificial Intelligence, San Francisco, USA, Apr. 2015, pp. 223-228.

[18] H. Zhang, W. Wu, and M. Yao, "Boundedness and convergence of batch back-propagation algorithm with penalty for feedforward neural networks," Neurocomputing, vol. 89, pp. 141–146, July 2012.

[19] W. Herzallah, H. Faris, and O. Adwan, "Feature engineering for detecting spammers on Twitter: Modelling and analysis," SAGE Journals. First published January 9, 2017.