Online Social Networks and Media 18 (2020) 100079

Contents lists available at ScienceDirect

Online Social Networks and Media

journal homepage: www.elsevier.com/locate/osnem

A deep learning model for Twitter spam detection

Zulfikar Alom[a], Barbara Carminati[b,*], Elena Ferrari[b]

[a] Department of Computer Science, Asian University for Women (AUW), 20-A, M. M. Ali Road, 4000, Chittagong, Bangladesh
[b] Dipartimento di Scienze Teoriche e Applicate (DiSTA), Università degli Studi dell'Insubria, Via J H Dunant, 3, 21100, Varese, VA, Italy

ARTICLE INFO

Article history:
Received 8 November 2019
Revised 21 April 2020
Accepted 23 April 2020

Keywords:
Spam detection
Social networks
Deep learning

ABSTRACT

Social networking platforms have become a popular way for Internet surfers to meet and interact. Twitter is one of the most popular social networking platforms where users can read the news, share ideas, discuss social issues, as well as stay in touch with friends and families. Due to its huge popularity, it has also become a target for spammers. Until now, researchers have developed many machine learning (ML) based methods for detecting spammers on Twitter. However, the available ML-based methods cannot efficiently detect spammers on Twitter due to possible data manipulations by spam users to avoid detection mechanisms. As an alternative to ML-based detection, in this paper, we present a new approach based on deep learning (DL) techniques. Our approach leverages both on tweet text as well as users' meta-data (e.g., age of an account, number of followings/followers, and so on) to detect spammers. We compare the performance of the proposed approach with five ML-based and two DL-based state of the art approaches on two different real-world datasets, showing a gain in performance when using our approach.

1. Introduction

In the last few years, Online Social Networks (OSNs), like Twitter, Facebook, and LinkedIn, become extremely popular communication tools [1,2]. As a result, users spend a great amount of time on OSNs, becoming friends with people they have known or have similar interests, sharing various types of content, such as news, events, personal problems, and so on. Twitter has become one of the most popular microblogging social network platforms, where users post messages of around 280 characters, known as *"tweet"*. It has been reported that Twitter has over 330 million monthly active users as of 2019, which posted over 500 million tweets every single day [3]. However, due to the huge popularity of Twitter, it also attracts the interest of cybercriminals (e.g., spammers) [4,5]. Spammers exploit the implicit trust relationships among users in order to achieve malicious aims, for instance, spreading malicious URLs within tweets, spreading rumors, sending unsolicited messages to other users, and so on [1].

As a result, many researchers and Twitter itself have proposed different spam detection techniques to make Twitter as a spam-free platform. For example, Twitter considers an account as a potential spam account if it has a high number of followings but a low number of followers [6]. Moreover, Twitter suspends accounts if they behave abnormally, such as frequently sending friend requests to whom they do not know, posting duplicate contents, and so on. At the same time, researchers have proposed many techniques to detect spam on Twitter (see Section 2 for more details). Most of the proposed approaches exploit machine learning (ML) algorithms that, once trained, are able to classify a user as spammer or non-spammer, based on a set of features related to his/her account or/and his/her tweets. Some of them (e.g., [4,7–11]) use account-based or text-based features, such as age of an account, number of followers/followings, number of tweets, number of URLs in a tweet, URLs to tweet ratio, and so on. However, by utilizing some tactics (such as mixing normal tweets, posting heterogeneous tweets, or buying followers from the third-party marketplace[1]) Twitter spammers can modify these features, making these mechanisms not always effective. For this reason, researchers come up with the idea of extracting some features from the social graph, resulting thus in hybrid approaches, that is, using both profile-based and content-based features (e.g., [1,2,12–15]). Graph-based schemes rely on more robust features that cannot be easily fabricated by malicious users, like distance and connectivity in the Twitter graph. However, since Twitter graph is tremendous in size, extracting these features from it requires a significant amount of time and resources, so that graph-based methods become very complex to be used in real-world scenarios.

* Corresponding author.
E-mail addresses: zulfikar.alom@auw.edu.bd (Z. Alom), barbara.carminati@uninsubria.it (B. Carminati), elena.ferrari@uninsubria.it (E. Ferrari).

[1] https://www.buyrealmarketing.com/buy-twitter-followers

https://doi.org/10.1016/j.osnem.2020.100079
2468-6964/

As alternative to ML-based solutions, recently deep learning (DL) approaches have started to be investigated [16,17], achieving promising accuracy results. This motivates us to follow this research direction and design a more accurate and robust DL-based model to detect Twitter spammers.

To achieve this goal, in this paper, we propose a new deep learning based approach that brings the nice benefit of being very fast, since it does not require any manual investigation to extract features, nor excessive computational resources, like other ML-based approaches.

Moreover, to understand how different types of features impact the classification process, we have designed two classifiers. The first is a *Text-based classifier* which only considers users' tweet text and exploits *Word2Vec* training method to learn tweet syntax. Then, we present a *Combined classifier* that considers both users' tweet text and meta-data. In particular, we have selected 9 easy to extract user/account's meta-data and text-based features, such as age of an account, number of followings/followers, number of tweets (see Section 4 for more details).

To assess our detection methods, we have selected two datasets: the popular social honeypot dataset [10], and the $1KS-10KN$ dataset [1]. Through the experiments, we show that our approach gives better performance than existing DL-based approaches and five of the ML-based state of the art approaches.

In summary, the main contributions of our work can be summarized as follows:

- we design a novel deep learning framework that has been proved to be powerful for spam account detection on Twitter;
- we demonstrate the importance of combining available meta-data (e.g., age of an account, number of followers/followings, and so on) for detecting spammers on Twitter;
- we compare our approach with the two DL-based approaches [16,17] and five ML-based state of the art approaches [10,11,18–20], showing that our approach gives better results on two different datasets;

The remainder of this paper is organized as follows. Section 2 presents related work. Section 3 explains the background on Convolution Neural Network (CNN), used in our approach. Section 4 explains our deep learning model for spammers detection. Section 5 illustrates experimental results, whereas Section 6 concludes the paper.

2. Related work

Spamming detection is not a new area of study. It has been studied for a decade. In the past, it mainly focused on email [21] and web spam detection [22], whereas over the last few years it has been an increasingly trending topic especially for online social networks.

Focusing on Twitter, literature presents many work exploiting users' features as training data for machine learning classifiers (e.g., [4,7–11]). For instance, Ala'M et al. [7] used four machine learning classifiers and some of the most common account-based and text-based features for detecting spammers on Twitter. Similarly, Ameen et al. [8] used four machine learning classification algorithms and 13 text-based features. Benevento et al. [4] compared two approaches for detecting spam profiles and spam tweets. Initially, they built their model to identify spam profiles based on account-based features. By using the Support Vector Machine (SVM) classifier their work achieves 84.5% accuracy. Then, they used both account-based and text-based features to classify tweets into spam and non-spam categories, by achieving 87.6% accuracy. Lee et al. [10] used 10 machine learning classifiers and two different datasets for spammers detection. Chen et al. [9] collected a large dataset of over more than 600 million public tweets. Then, they extracted 12 lightweight features[2] and used them into six machine learning algorithms to classify tweets into spam and non-spam categories. In their experiments, Random forest (RF) gives the highest 93.6% f1-score. Similarly, Stringhini et al. [11] used 6 lightweight features and different machine learning algorithms.

Although these approaches present the nice benefit of an easy extraction of tweet text-based and account-based features, they all suffer from the same drawback. Indeed, spammers can avert these features to avoid being detected. To address this problem, researchers have leveraged on the social graph to come up with more robust features. For example, Hai Wang et al. [15] leveraged on the followings and followers relationships. They extracted around $25K$ Twitter users, and 20 recent tweets for each user, along with $49M$ friend/follower relationships. To assess their detection method, they used four different classifiers i.e., Decision Tree (DT), Neural Network (NN), Naive Bayes (NB), and SVM. In the experiments, NB classifier gives the best performance: 91.7% precision and 91.7% f1-score. Gao et al. [13] also detected spammers according to the users' social degree, interaction history, cluster size and so on.

Fazil et al. [2] proposed an hybrid framework, exploiting users' meta-data, in addition to graph-based, and tweet text-based features to detect spammers on Twitter. They used 19 features and three popular machine learning algorithms (i.e., RF, DT, NB) for classifying users into spammers and non-spammers. In their experiments, RF classifier gives the best performance, achieving 97.9% f1-score. Likewise, Yang et al. [1] used four feature sets, namely, account-based, text-based, graph-based, and automation-based features (e.g., the usage frequency related to Twitter API) and three machine learning classification algorithms for classifying users into spammers and non-spammers. Their experiments achieve 90% f1-score.

In the literature review, we have found few papers that, like our approach, are based on deep learning (e.g., [16,17,23–25]). [16] used *Word2Vec* deep learning technique and further *Doc2Vec* training model to assign a multi-dimensional vector to a tweet. After that, it is passed through different classifiers for predicting whether the tweet is a spam or non-spam. They compare the results of several classifiers and shows that Random Forest (RF) gives better results in terms of precision and accuracy (94%). In contrast, we use only a *Word2Vec* deep learning tool for representing a tweet in a high dimensional matrix and then we use a convolution neural network (CNN) for extracting the most accurate and relevant features vector of each tweet. In addition, we also consider Twitter users' meta-data information combined with tweet text-based features. More importantly, we get better results than their approach in terms of accuracy and f1-score (see Section 5).

Xinbo Bon et al. [17] proposed a new feature engineering mechanism based on a deep neural network using Bidirectional Long Short-Term Memory (Bi-LSTM) method. More precisely, they train a deep neural network using labeled dataset and extract features from the hidden layers of it to represent tweets. In Section 5, we compare their approach with ours, showing that we achieve better results. Similarly, Tao et al. [23] applied deep learning-based operations on concatenated word vectors in the convolutional layer. Mou et al. [24] applied convolutional models on sentences having hierarchical structures. Tai et al. [25] changed the LSTM structure to model tree-structured topologies by sequentially stacking both CNN and LSTM and achieved promising results for semantic sentence modeling.

As a summary, Table 1 lists the used features and categories of the different feature based approaches described so far. We

[2] A feature that is relatively easy to extract from the Twitter user's profile is called a lightweight feature.

Table 1
Summary of the features used by the surveyed approaches along with their categories.

Features	Category	Existing Approaches														Our approach
		[8]	[9]	[15]	[18]	[7]	[11]	[14]	[1]	[10]	[13]	[12]	[2]	[19]	[20]	
Age of an account	Profile	✓	✓						✓	✓		✓		✓		✓
Number of followers	Profile	✓	✓	✓	✓			✓				✓		✓		✓
Number of following/friends	Profile	✓	✓	✓	✓		✓	✓	✓			✓		✓		✓
Number of favorites	Profile	✓	✓													✓
Number of lists	Profile	✓	✓									✓				
Number of tweets	Profile	✓	✓				✓		✓					✓	✓	✓
Reputation score	Profile			✓	✓			✓				✓	✓			✓
Following to followers ratio (FIFO)	Profile					✓	✓		✓	✓		✓		✓		✓
Followers ratio	Profile												✓	✓		
Following ratio	Profile								✓					✓		
Followers' following to followers ratio	Profile												✓			
Followers based reputation Score	Profile												✓			
Length of profile description	Profile											✓		✓		
Length of profile name	Profile											✓		✓		
Presence/absence of profile image	Profile					✓						✓				
Number of tweets posted per day	Profile					✓			✓	✓		✓		✓		
Average tweets length	Content											✓				
Number of hashtags	Content	✓	✓	✓	✓							✓			✓	
Number of hashtags per tweet	Content											✓	✓		✓	
Number of URL links	Content	✓	✓	✓	✓			✓							✓	
Number of URL links per tweet	Content		✓				✓		✓	✓	✓	✓	✓	✓	✓	
Number of unique URL links per tweet	Content								✓	✓	✓	✓	✓	✓		
Number of user mentions	Content	✓	✓	✓	✓			✓							✓	
Number of user mentions per tweet	Content									✓			✓	✓	✓	
Number of unique user mentions per tweet	Content									✓		✓	✓	✓		
Number of spam words	Content					✓									✓	
Number of spam words per tweet	Content														✓	
Number of words	Content														✓	
Number of re-tweets	Content	✓	✓												✓	
Number of re-tweets per tweet	Content											✓	✓			
Number of user replies	Content			✓	✓			✓								
Number of user replies per tweet	Content											✓				
Number of characters per tweet	Content	✓	✓													✓
Number of digits per tweet	Content	✓	✓													✓
Number of keywords per tweet	Content							✓								
Tweet similarity	Content			✓	✓		✓		✓	✓		✓		✓		
Zip compression ratio of posted tweets	Content													✓		
Text to link ratio	Content					✓										
Repeated words	Content					✓										
Content and hashtags similarity ratio	Content												✓			
Comments ratio	Content					✓										
Average spam post count	Content											✓				
Different descriptions from tweets	Content					✓										
Different following interests from tweets	Content					✓										
Friends' name similarity	Content						✓									
Tweet time pattern	Content					✓		✓			✓	✓				
Bi-directional link ratio	Graph								✓	✓						
Betweenness centrality	Graph								✓							
Clustering co-efficient	Graph								✓				✓			
Sender social degree	Graph										✓					
Interaction history	Graph										✓					
Cluster size	Graph										✓					
Community based reputation	Graph												✓			
Community based cluster coefficient	Graph												✓			
Average neighbors' followers	Graph								✓							
Average neighbors' tweets	Graph								✓							
Following to median neighbors' followers	Graph								✓							
API ratio	Automation								✓							
API URL ratio	Automation								✓							
API tweet similarity	Automation								✓							
Automated tweet ratio	Automation												✓			
Automated tweet URL ratio	Automation												✓			
Automated tweet similarity	Automation												✓			
Tweet time standard deviation	Time												✓			
Tweet time interval standard deviation	Time												✓			
Standard deviation of numerical IDs of followings	Time													✓		
Standard deviation of numerical IDs of followers	Time													✓		
Mean time between tweets	Time														✓	
Standard deviation time between tweets	Time														✓	
Idle duration time between tweets	Time														✓	
Change rate of number of followings	Time													✓		

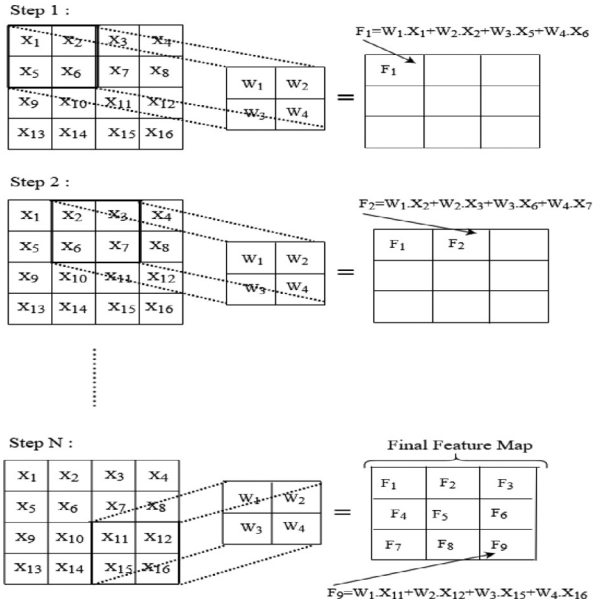

Fig. 1. Steps of convolution operations.

classify Twitter features into five categories, namely: profile-based (i.e., features based on users' accounts), content-based (i.e., features based on users' tweet text), graph-based (i.e., features based on Twitter users' community), automation-based (the usage frequency related to Twitter API), and time-based (e.g., the speed a Twitter user follows others).

3. Convolution neural networks

Since the proposed DL-based spam detection approach exploits Convolution Neural Networks (CNNs), in this section we give a very brief introduction to them. CNN is one of the most widely used artificial neural networks in DL, whose primarily usage is for image classification, object detection, and natural language processing (NLP) [26–29]. Typically, a CNN consists of three layers [26], namely: convolution (CONV) layer, POOL layer, and fully connected (FC) layer.

CONV layer: this layer is the main building block of CNN, whose primary objective is to extract features from the input data. Convolution is applied over the input data using the convolution filter to produce a feature map. As an example, let us consider Fig. 1, where: $X_1 \ldots X_{16}$ is a matrix of 16 numbers representing an image or text; the set of parameters (aka weights) $W_1 \ldots W_4$ is called the kernel or filter. During the training phase, CNN automatically learns the values of its filters based on the task that one wants to perform. For instance, in image classification, filters are used to detect the shapes of eyes, face, and so on. In our experiments, we use 100 of bi-gram, tri-gram, and four-gram filters.

The convolution operation computes a dot product[3] between the filter and a local region of the input matrix. It is important to note that the local region size would have exactly the same dimension as the filter. In Fig. 1, we choose a 2X2 kernel and local region size, but the selection of kernel size is flexible. The outcome of this operation is a single number which is called the feature map. Then, we shift the filter to the next receptive field (aka local region) in the same input matrix by a Stride[4] and compute the dot product again between the new receptive field and the same filter. The process is repeated until we go through the entire input matrix. Fig. 1 shows the performed steps.

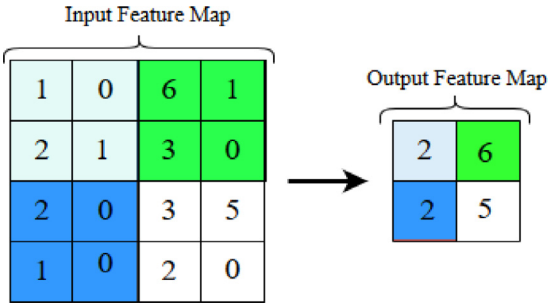

Fig. 2. Max pooling operations.

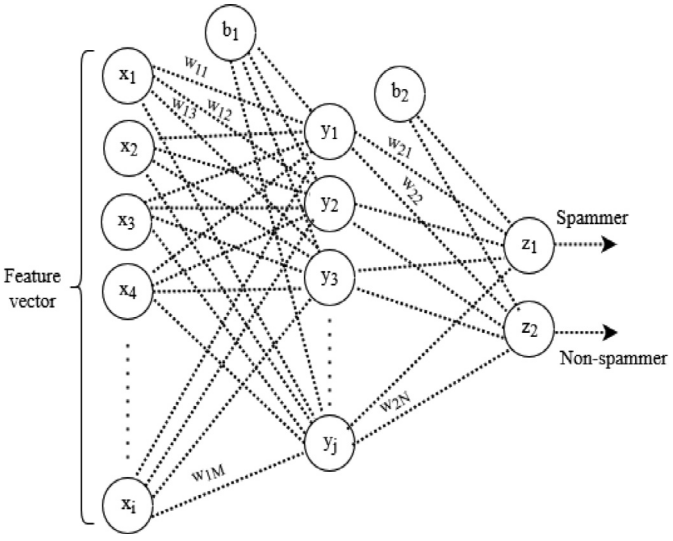

Fig. 3. Architecture of the FC layer.

POOL layer: this layer performs a function to reduce the dimensionality of the input. Since it reduces the number of parameters, it brings the benefits of shortening the training time and control the overfitting.[5] There are different types of functions, such as Max pooling, average pooling, and so on. However, Max pooling is the most widely used, which only takes the maximum value in the window. Let us consider the example in Fig. 2, where the pool layer takes the 4X4 feature map as input from the CONV layer and performs max pooling function with 2X2 windows size, and stride of 2 over it. For each of the windows, max pooling takes the maximum value of the 4 number, consequently, it generates a 2X2 feature output, which will send to the fully connected layer.

FC layer: the fully connected (FC) layer is used to wrap up the CNN architecture. Since the FC layer expects a vector as input, the output of the pooling layer is previously converted into a 1*D* feature vector, this operation is called flattening. The basic architecture of the FC layer is shown in Fig. 3, where $x_1 \ldots x_i$ is the feature vector, $y_1 \ldots y_j$ are neurons, and $z_1 \ldots z_2$ is the output class (i.e., spammer, non-spammer). Each neuron receives several inputs, passes them through to an activation function (Y), and produces an output vector whose dimension is the same as the number of possible output classes.

[3] Dot product refers to the element-by-element product or element-wise multiplication.

[4] In the context of CNN, stride means the amount by which the filter shifts horizontally and vertically while performing convolution operation.

[5] Overfitting is the case where the predictor model fits perfectly on the training data but does badly on the test data.

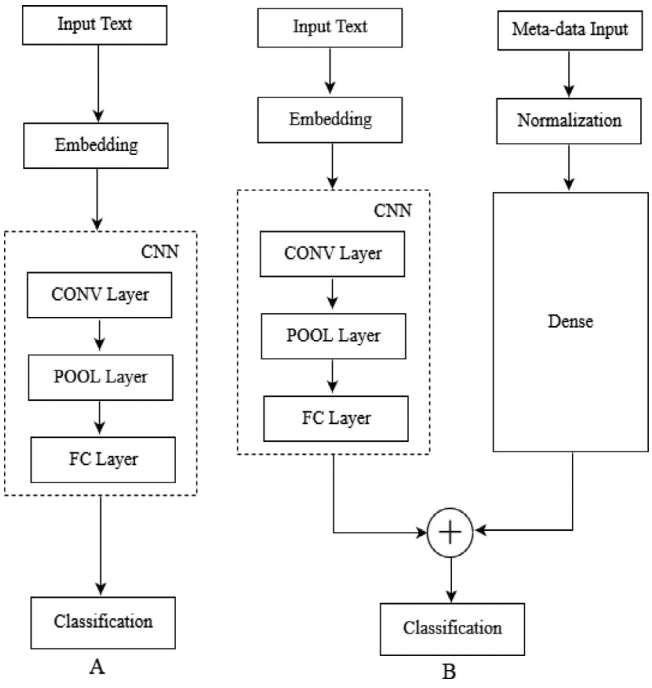

Fig. 4. (A) Text-based classifier; (B) Combined classifier;.

4. Proposed methodology

In this section, we present the overall idea our Twitter spam detection method. As introduced in Section 1, we have developed two different classifiers, namely *Text-based* and *Combined*, both exploiting CNNs by considering different set of features.

We opt for CNNs since it has been demonstrated their abilities in finding complex patterns, as well as their good computational time [30,31]. As depicted in Fig. 4, both the proposed classifiers consist of several steps, which will be detailed in what follows.

4.1. Text-based classifier

This model consists of three major steps (see Fig. 4(A)), which are described in what follows.

Embedding: this step transforms each tweet before sending it to the CNN. Each tweet is mapped to a matrix, where each word in the tweet is represented as a high dimensional row vector (see Fig. 5). We use the 200 dimensional word embedding, as it gives the higher accuracy in text classification [32]. This means that every word of the tweet is represented by a 200 dimensional vector. In particular, we use *Word2Vec* [33], which does not rely on occurrence counts. In contrast, word vectors are generated based on the surrounding context.[6] *Word2Vec* can utilize two models to produce the embedding of words, namely, skip-gram, and continuous bag-of-words (CBOW). They both take the text and generate the row vector by predicting its surrounding context (skip-gram model), or predicting the word given its surrounding context (CBOW model), using gradient descent algorithm.

Moreover, since tweet text's length might vary, we adopt a padding technique to ensure that all tweet matrices have the same length. We set a zero-pad at the beginning, if a sentence length is shorter than the maximum length. Once the embedding layer terminates, the high dimensional matrix is feed to the CNN step.

CNN: as described in Section 3, CNN is made up of neurons with learnable weights and biases. It consists of three layers: CONV layer, POOL layer, and FC layer. The *CONV layer* takes the input data and computes a dot product between the filter and the input data. In our model, instead of using only one bi-gram filter, we use also tri-gram and four-gram filters, as shown in Fig. 5. The *POOL layer* performs a down-sampling operation, which means that it reduces the dimension size of the input data. Finally, the *FC layer* contains 256 neurons with the activation function (Y) used for generating the output vector, whose dimension is the same as the number of possible output classes. In our implementation, we experimented different CNN architectural models and we ended up using Zhang's model proposed in [29], because it gives better performance [31] than other CNN architectures. After performing convolutional and max-pooling operations, this model simply concatenates max pooled results from each of bi-gram, tri-gram, and four-gram filters, then it sends them into the FC layer.[7] This layer generates the output vector by using activation function (Y), having exactly two output classes (i.e., spammer, non-spammer).

Classification: this step uses a Softmax function for predicting the class score. This function is a very popular function for estimating the probabilities of events. It can be defined as follows: $\sigma(v)_j = \frac{\exp^{v_j}}{\sum_{z=1}^{Z} \exp^{v_z}}$, for $j = 1 \ldots Z$; where v is the vector of inputs, and Z represents the number of possible outcomes. Since we have 2 possible classes (i.e., spam and non-spam), j contains 2 elements. The output of this function is always in the range [0,1], where the sum of the output values is equal to 1. It will classify input to class 1 when the output of the function is close to 1 (i.e., output > 0.5), and classify to class 2, when the output is close to 0 (i.e., output ≤ 0.5).

Example 1. Consider a user U and his/her tweet $t =$ *"I made* \$1000 *today check out"*. First, the embedding step transforms t to a high-dimensional matrix. We use *Word2Vec* embedding with 200 dimension, however, for better understanding, in this example we use a 5 dimensional word embedding. Let us consider that 'I' maps to [0, 1, 2, 3, 1];[8] likewise, 'made' maps to [1, 0, 1, 1, 0], and so on. Therefore, t is represented by the following 6X5 Tweet Matrix TM.

$$TM = \begin{bmatrix} 0 & 1 & 2 & 3 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 2 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 2 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{bmatrix}$$

Next, this matrix is sent to the CNN step, to generate a feature vector. In the convolution layer, we use 100 different bi-gram, tri-gram, and four-gram convolution filters whose weights are randomly initialized. However, for the sake of clarity, in this example, we consider only one bi-gram, one tri-gram, and one four-gram filter. In particular, the following is the used bi-gram convolution filter (BG):

$$BG = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

The value of this filter is multiplied with TM (aka computing element-wise multiplications). These multiplications are all summed up and return a single number. This process is repeated for every location on the input matrix. Next step is moving the filter down by 1 unit and so on. After performing this operation over

[6] Given a tweet t and word w_i in t, the surrounding context is the preceding and following words of w_i, i.e., w_{i-2}, w_{i-1}, w_{i+1}, w_{i+2} for 2 windows surrounding context.

[7] Note that here flattering is not needed as the result of max-pooling is not a matrix.

[8] The values are provided by the *Word2Vec* embedding tool. Interested readers can refer to [34] for more details.

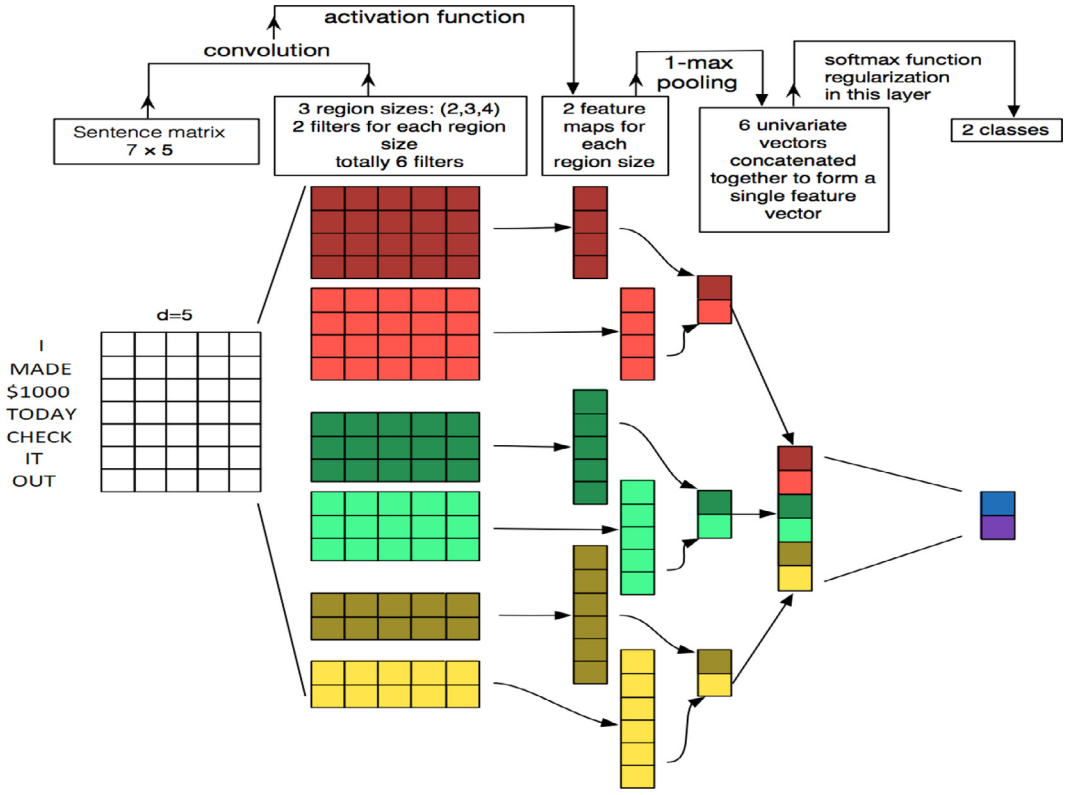

Fig. 5. CNN architectural model for sentence classification [29].

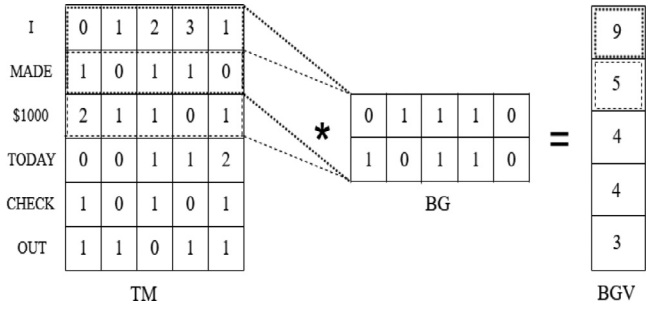

Fig. 6. Operations of the CONV layer

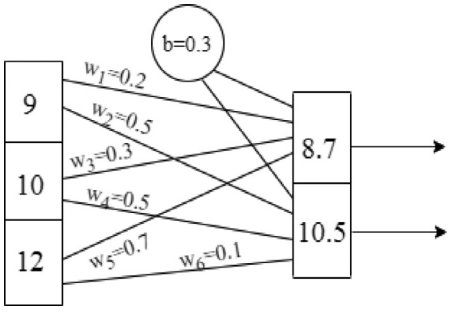

Fig. 7. Operations of the fully connected (FC) layer.

all locations of the input matrix, we get the column vector (*BGV*) depicted in Fig. 6.

Let us assume the following tri-gram convolution filter (TG):

$$TG = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

It generates the column vector (*TGV*):

$$TGV = \begin{bmatrix} 10 \\ 6 \\ 4 \\ 6 \end{bmatrix}$$

whereas the following four-gram convolution filter (FG):

$$FG = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

generates the following column vector (*FGV*):

$$FGV = \begin{bmatrix} 12 \\ 8 \\ 7 \end{bmatrix}$$

Then, the max-pooling operation selects the maximum value of each column vector. Therefore, $MAX(BGV) = 9$, $MAX(TGV) = 10$, and $MAX(FGV) = 12$.

Therefore, the obtained tweet vector is $TV = [9, 10, 12]$, which is sent to the FC layer. As depicted in Fig. 7, this generates the output vector by using activation function (Y), having only two possible output classes.

As shown in Fig. 7, we consider the following weights: $w_1 = 0.2$, $w_2 = 0.5$, $w_3 = 0.3$, $w_4 = 0.5$, $w_5 = 0.7$, $w_6 = 0.1$, and bias $b = 0.3$. Therefore:

$$\begin{aligned} y_1 &= \sum wx + b \\ &= w_1x_1 + w_3x_2 + w_5x_3 + b \\ &= 0.2 * 9 + 0.3 * 10 + 0.7 * 12 + 0.3 \\ &= 13.5 \end{aligned}$$

Similarly:

$$y_2 = \sum wx + b$$
$$= w_2x_1 + w_4x_2 + w_6x_3 + b$$
$$= 0.5 * 9 + 0.5 * 10 + 0.1 * 12 + 0.3$$
$$= 10.5$$

Finally, the vector [13.5 10.5] passes through the softmax function in the classification step. It estimates the class probabilities of the tweet. Hence, $\sigma(spammer) = \frac{\exp^{13.5}}{\exp^{13.5}+\exp^{10.5}} = 0.95$; and, $\sigma(non-spammer) = \frac{\exp^{10.5}}{\exp^{13.5}+\exp^{10.5}} = 0.05$. Therefore, user U who posted the tweet t is classified as spammer.

4.2. Combined classifier

To demonstrate the importance of combining meta-data related to users' account for detecting spammers on Twitter, we consider also a combined classifier. This exploits both the text-based classifier defined in Section 4.1, as well as a simple neural network on users' meta-data. Then, to learn whether a user is a spammer or not, the Combined classifier merges the results obtained by both the text and meta-data classifiers (see Fig. 4(B)), as explained in what follows.

Meta-data classifier: this classifier considers users' meta-data as input. In particular, we consider the following meta-data: age of an account, number of followers, number of followings/friends, number of favorites, number of tweets, reputation score, following to followers ratio, number of characters per tweet, and number of digits per tweet. For this classifier, we use a simple neural network of several fully connected layers. The proposed classifier consists of two major steps (i.e., Normalization, and Dense) briefly described in what follows.

- **Normalization.** Neural networks give better performance (e.g., faster learning and higher accuracy) when the input data have 0 as mean and 1 as variance [35]. As such, as a first step, we normalize the input meta-data. To do this, first, we calculate the mean value and standard deviation of each feature. Next, we subtract the mean value from each feature. Finally, we divide the value of each feature by its standard deviation. Mathematically: $x' = \frac{x-\bar{x}}{\sigma}$, where x is the original feature vector, $\bar{x}$ is the mean value of that feature vector, and σ is its standard deviation.
- **Dense.** In this step, we use a simple neural network of several fully connected layers to generate more accurate and extended (i.e., 256 dimensional) feature vectors. In our experiments, we tested different architectures and we ended up using 5 layers of size 512, 256, 128, 64, and 32, as this gives better results among all the tested configurations. Moreover, on top of all these 5 layers, we add one additional layer, which ensures that the meta-data classifier output has the same dimension as the one of the tweet text classifier, so as to be able to fuse them together.
  We recall that for the text-based classifier we exploit a CNN with 256 neurons, so it produces a 256 dimensional tweet features vector. Likewise, the meta-data classifier generates a 256 dimensional features vector. As shown in Fig. 8, we use a 6 layers architecture and each layer has a different amount of neurons, for instance, first hidden layer (HL-1) has 512 neurons, second hidden layer (HL-2) has 256 neurons, and so on. Each neuron in the hidden layer takes the entire output of the previous layer as input. For instance, in our case, each neuron in the HL-1 takes the users' 9 normalized meta-data features as input. Similarly, each neuron in the HL-2 takes the HL-1's 512 neuron value as input.

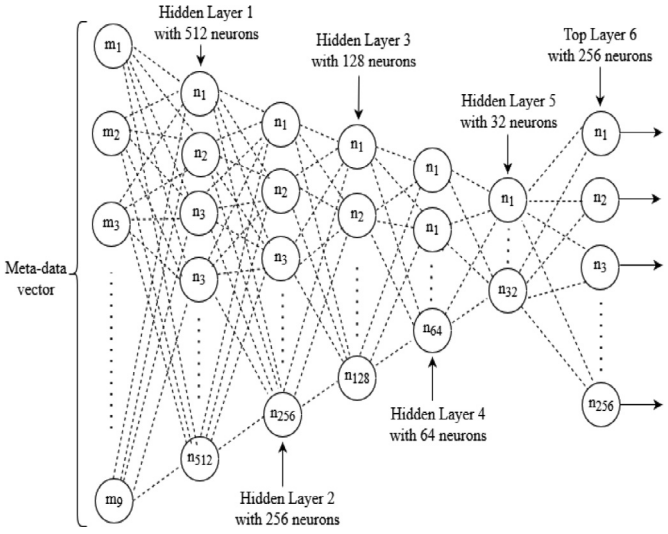

Fig. 8. Network architecture of the Dense step.

Concatenation and classification: we concatenate the tweet text and meta-data classifier output and send it to the classification step, which is a fully connected layer with softmax activation function. It gives the final prediction whether the user is a spammer or non-spammer.

To better understand how the combined approach works, let us consider the following example.

Example 2. Suppose a user U has the following set of meta-data: age of the account = 90 days, number of followers = 150, number of followings/friends = 99, number of favorites = 22, number of tweets = 25, reputation score = 0.60, following to followers ratio = 0.66, number of characters per tweet = 101, and number of digits per tweet = 9. Moreover, let us assume that U posts the following tweet t = *"I made* $1000 *today check out"*. In this combined approach, we merge the output of the two classifiers to make joint feature vector before sending it into the classification step.

As described in Example 1, for t the text-based classifier generates a 3 dimensional tweet feature vector TV = [9 10 12].

The meta-data classifier considers 9 meta-data information of Twitter users, which can be represented as a vector MDI = [90 150 99 22 25 0.60 0.66 101 9]. The meta-data classifier, first normalizes the MDI, by passing through the normalization step. Let us assume that the normalized MDI is MDI' = [0.1 0.15 0.2 0.22 0.25 0.06 0.16 0.01 0.09]. Next, it sends MDI' into the dense step. This step generates the user's meta-data features vector UMD, which has the same dimension as the tweet-text classifier's features vector.

As shown in Fig. 9, we assume that the adjusted weight values are: 0.1 for $w_1 \dots w_9$, 0.2 for $w_{10} \dots w_{18}$, 0.3 for $w_{19} \dots w_{27}$, with bias $b = 0.1$. Therefore, by using activation function (Y) we get:

$$y_1 = \sum wx + b$$
$$= w_1x_1 + w_4x_2 + w_7x_3 + w_{10}x_4 + w_{13}x_5 + w_{16}x_6$$
$$+ w_{19}x_7 + w_{22}x_8 + w_{25}x_8 + b$$
$$= 0.1 * 0.1 + 0.1 * 0.15 + 0.1 * 0.2 + 0.2 * 0.22$$
$$+ 0.2 * 0.25 + 0.2 * 0.06 + 0.3 * 0.16 + 0.3 * 0.01$$
$$+ 0.3 * 0.09 + 0.1$$
$$= 0.33$$

Likewise, $y_2 = \sum wx + b = 0.32$, and $y_3 = 0.33$.

Hence, the final meta-data feature vector is UMD = [0.33 0.32 0.33]. Next, we combine the two classifier's output and

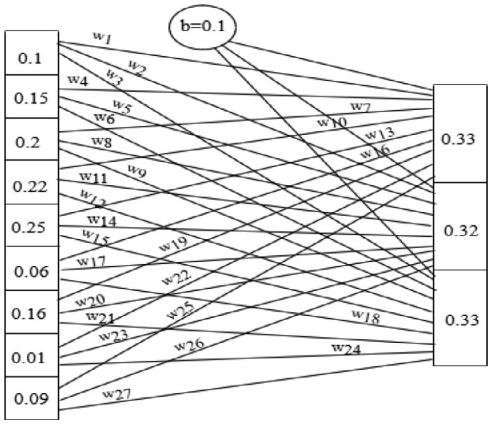

Fig. 9. Operations of the meta-data classifier.

generate a combined feature vector $CFV = [9\ 10\ 12\ 0.33\ 0.32\ 0.33]$, which is then sent to the classification step.

The classification step, like fully connected (FC) layer (see Fig. 7), uses the activation function and generates the output vector whose dimension is the same as the number of output classes. Since we have only two output classes (i.e., spammer and non-spammer), the output vector has 2 values. Hence, by using activation function Y, the output vector would be [6.02 5.17]. Finally, the vector is passed through the softmax function to estimates the class probabilities of the user. $\sigma(spammer) = \frac{\exp^{6.02}}{\exp^{6.02}+\exp^{5.17}} = 0.71$; and, $\sigma(non-spammer) = \frac{\exp^{5.17}}{\exp^{6.02}+\exp^{5.17}} = 0.29$. Hence, based on the example, we can say that the user U is a spammer user.

5. Experimental results

In this section, we present the results of the experimental evaluation we carried out to show the effectiveness of the proposed approach. Through the experiments on two real-world Twitter datasets, we try to find answers to the following two questions:

- how effective is the proposed approach compared with existing machine learning and deep learning based state of the art approaches?
- How effective is the proposed architecture compared with simpler architectures?

In order to answer these questions, first, we introduce the two real-world Twitter datasets we have used in our experiments. Then, we present the performance evaluation in terms of accuracy, precision, recall, and f1-score, as well as an analysis and discussion of the results.

5.1. Datasets

The first dataset we used in our experiments is the Twitter Social Honeypot dataset [10], in which users have been already classified as spammers and non-spammers based on tweet content, user behavioral, and topological features. This dataset contains 22,223 spammers and 19,276 non-spammer users, that were captured during an eight-month period in 2010. At this aim, first, authors created 60 social honeypot accounts on Twitter, whose purpose were to act as Twitter users, and report back what accounts follow or interact with them. They sent random messages and perform many activities, in this way they attracted spammers. Then, they used Expectation-Maximization (EM) clustering algorithm to find groups of harvested users with similar appearances/behaviors, and assign a probability distribution about the clusters to each harvested user's account. Then, they manually labeled harvested users into spammers and non-spammers. From this dataset (we name this dataset as Dataset I in what follows), we randomly selected 2,461 users, 1,632 spammers and 829 non-spammers. Since the dataset is quite old, for further validation, we have manually checked 30% of collected data and found that all checked data are still labeled correctly. However, in Dataset I, most of the users do not have their complete profile information. Therefore, we consider only those users who have complete profile information, such as lists of followers/followings, etc. In addition, we also excluded those users who posted tweets in non-English languages, since we have used word embedding (i.e., *Word2Vec*) which is based on the English language only.

Table 2
Characteristics of the used datasets.

Dataset	Property	Value
I	Number of Twitter users	2461
	Number of spam users	1632
	Number of non-spam users	829
	Number of tweets	2461
II	Number of Twitter users	1125
	Number of spam users	596
	Number of non-spam users	529
	Number of tweets	1125

The second dataset is the Twitter $1KS - 10KN$ dataset (we name this dataset as Dataset II in what follows), provided by [1].[9] To create this dataset, the authors recursively crawled Twitter accounts in multiple rounds. First, they collected 20 Twitter seed accounts, then they collected followers and followings of all seed accounts. This crawling process has been repeated until they reached the target number of users. The authors have mainly focused on those spammers who post malicious URLs. They defined a tweet that contains at least one malicious or phishing URL as a spam tweet, and a user whose spam ratio is higher than 10% as a spammer. This dataset contains a total of 11,000 labeled users, 1,000 spammers and 10,000 non-spammer users. In this dataset, users post tweets with various languages, such as Spanish, Italian, etc. Thus, we excluded users who post tweets that are not written in English. Therefore, we finally obtain a dataset with 1,125 users, 529 legitimate users and 596 non-legitimate users. The basic characteristics of the datasets used in our experiments are shown in Table 2.

5.2. Experimental setup

In the experiments, we used Keras[10] deep learning tool, which uses Tensorflow[11] as back-end for the deep learning model implementation. Moreover, for combining our two classifiers (i.e., tweet text classifier and meta-data classifier) into a single one, we used function API [36], which is a way for defining complex models, such as models with multiple inputs and outputs. Finally, we run the experiments on Linux OS (i.e., Ubuntu 18.04) with 3.00 GHz Intel Core i5 processor and 8 GB of RAM.

5.2.1. Parameters setting

We trained the models using Adam [37] optimization algorithm, which combines the two extensions (i.e., Adaptive Gradient Algorithm (AdaGrad) [38], and Root Mean Square Propagation (RMSProp) [39]) of stochastic gradient descent, with mini-batch size

[9] http://faculty.cse.tamu.edu/guofei/research_release.html

[10] Keras is an open source neural network library written in Python, which is designed for fast experimentation with deep neural networks. https://keras.io

[11] TensorFlow is an open-source software library for data flow programming across a range of tasks, which is used for different machine learning applications, such as neural networks. https://www.tensorflow.org

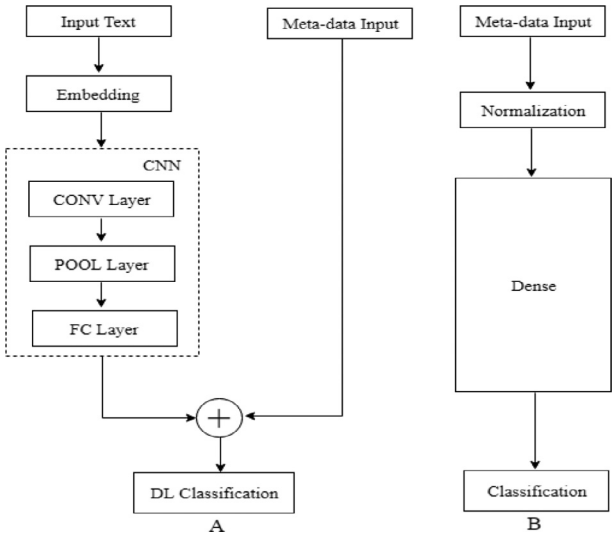

Fig. 10. (A) Simpler combined approach; (B) Meta-data approach;.

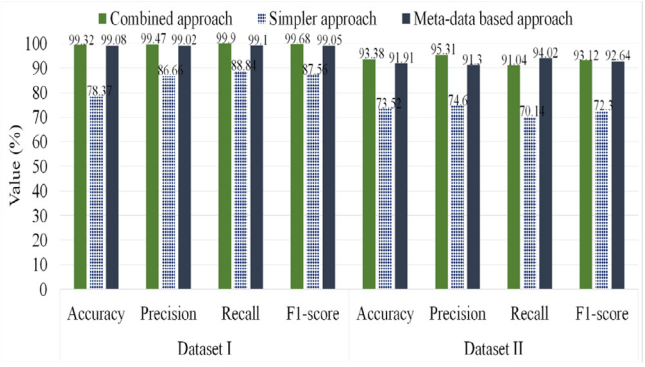

Fig. 11. Comparison among the proposed approach and a simpler approach and meta-data approach.

of 32 examples, and learning rate of 0.001. Moreover, we used *ReLUs* activation functions, and we kept an equal learning rate for all layers.

5.3. Performance metrics

We used four standard metrics, namely accuracy, precision, recall, and f1-score. Accuracy (A) is ratio of the total number of correctly classified instances of both classes over the total number of all instances in the dataset. Precision (P) refers to the ratio of the number of correctly classified instances to the total number of instances. Recall (R) defines the ratio of the number of instances correctly classified to the total number of predicted instances. F1-score (F1) is measured as a weighted average of the precision and recall, defined as $F1 = \frac{2P*R}{(P+R)}$.

5.4. Evaluation

In this section, we evaluate our approach by comparing it with the existing state of the art approaches. First, we compare it with existing ML-based and DL-based approaches. Then, we compare it with a more simplistic alternative architecture.

5.4.1. Comparison with state of the art approaches

We compare the performance of the proposed approach with five existing machine learning based state of the art approaches, namely: (A) [11], which used profile/account and content-based features; (B) [18], which also used profile and content-based features; (C) [10], which used profile, content, and graph-based features; (D) [19], which used profile, content, and timing-based features; (E) [20], which used profile, content, and time-based features (see Table 1), and two deep learning based approaches: (F) [16] and (G) [17].

For performance comparison, we have only selected those approaches which provided comparable and competitive results. More precisely, we have selected *A, B, C, D, F* and *G* for comparison with our model using Dataset I, because, from this dataset, it is possible to extract all of the features they considered. For the same reason, for Dataset II, we have selected *A, B, E, F* and *G* for comparison with our model.

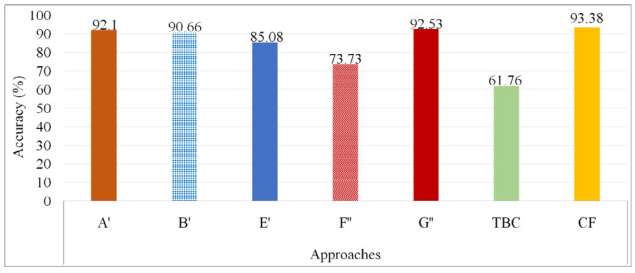

Figure 17: Accuracy

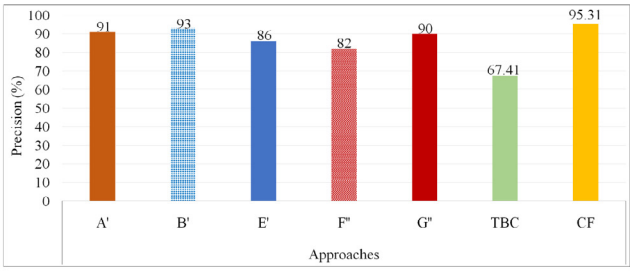

Figure 18: Precision

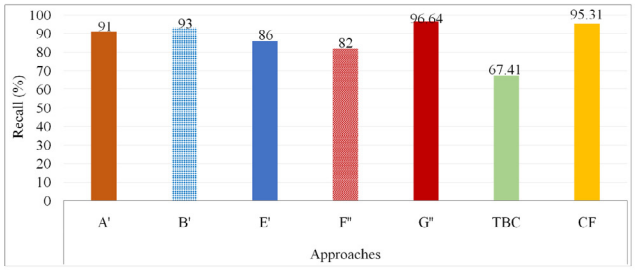

Figure 19: Recall

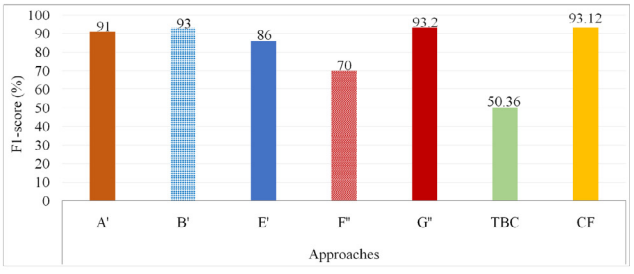

Figure 20: F1-score

Fig. 16. Performance comparison results (accuracy, precision, recall, and f1-score) over Dataset I should consist of figures 12, 13, 14, and 15.

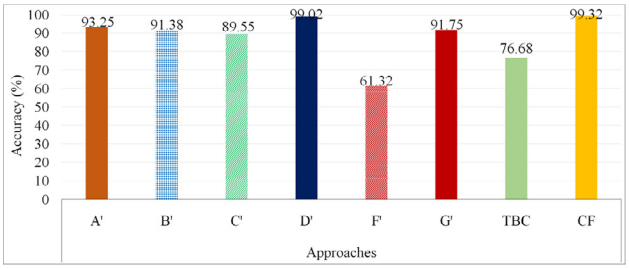

Figure 12: Accuracy

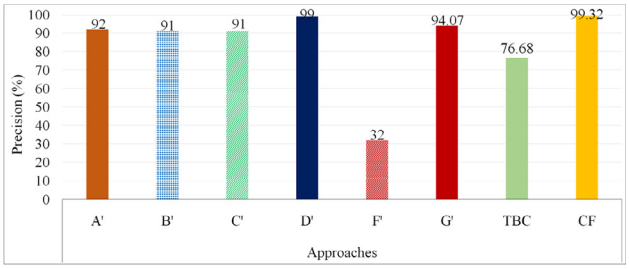

Figure 13: Precision

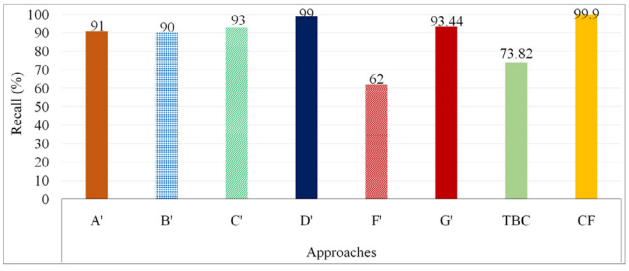

Figure 14: Recall

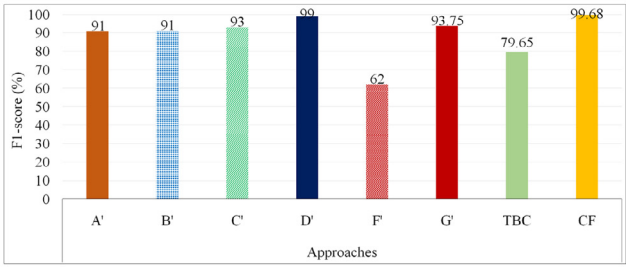

Figure 15: F1-score

Fig. 21. Performance comparison results (accuracy, precision, recall, and f1-score) over Dataset II should consist of figures 17, 18, 19, and 20.

Table 3 shows the accuracy, precision, recall, and f1-score of the considered machine learning and deep learning based approaches vs our approach, when four different classifiers, namely: Decision Tree (DT), Random Forest (RF), Logistic Regression (LR), and Support Vector Machine (SVM) are used, over the two datasets. In the experiments, we have used 10 fold cross validation approach. The best results are highlighted in bold. It can be observed that, the existing state of the art approaches give better performance with LR and RF over Dataset I and II, respectively. By using Dataset I and *RF, A* gives accuracy 93.25%, *B* gives accuracy 91.38%, *C* gives accuracy 89.55%, whereas *D* with LR classifier gives accuracy 99.02%, *F* with SVM classifier gives accuracy 61.32%, and *G* with LR classifier gives accuracy 91.75%. Likewise, by using Dataset II, approaches *A, B, E, F,* and *G* with classifier RF give the highest accuracy 92.1%, 90.66%, 85.08%, 67.25%, and 92.08%, respectively.

In this experiment, we compare the performance of the our approach with the existing approaches.

From Figs. 16 and 21, it can be observed that our combined approach outperforms all the others, namely A + RF, B + RF, C + RF, D + LR, E + RF, F + RF, F + SVM, G + LR, and G + SVM. Our text-based classifier is labelled as *TBC*, whereas the combined classifier is labelled as *CF*. Moreover, we label A + RF as A', B + RF as B', C + RF as C', D + LR as D', and E + RF as E', F + RF as F', G + LR as G', F + SVM as F'', and G + SVM as G''.

Over Dataset I, we can see that the f1-score of our approach (i.e., *CF*) is 99.68%, as shown in Fig. 12d. It gives the highest f1-score than other approaches. Similarly, in terms of accuracy *CF* gives better results (i.e., reaching 99.32%), as shown in Fig. 12a. Likewise, the CF's precision and recall value is 99.47% and 99.9%, respectively, which are also greater than those of the other considered approaches, as shown in Fig. 12b and c. Over Dataset II, from Fig. 13d, it can be seen that the f1-score of *CF* is 93.12%, which is greater than the ones of all the other considered approaches (i.e., *A, B, E, F,* and *G*). Similarly, from Fig. 13a to c, we see that *CF* gives accuracy, precision, and recall value 93.38%, 95.31%, and 91.04%, respectively.

However, from Figs. 12 and 13 it can be observed that, our other approach *TBC* (i.e., text-based classifier) has not performed well enough as we have expected. It gives only f1-score 79.65% in Dataset I and 50.36% in Dataset II. Indeed, to evade text-based detection features (i.e., tweet similarity, duplicate tweet count, and so on.), spammers mainly utilize the tactics of mixing normal tweets and posting heterogeneous tweets to act as the legitimate users. Consequently, using only *TBC* has not provided good results.

According to the performance evaluation discussed above, we can conclude that our proposed combined classifier approach is more effective to detect Twitter spammers than the other considered DL-based and ML-based state of the art approaches.

5.4.2. Comparison with a simpler alternative architecture

As previously presented, our proposed combined approach gives better performance than all considered state of the art approaches. However, we have run a further set of experiments to see whether we can have the same gain in the performance if we change the proposed combined approach by making it simpler. More precisely, we just combine users' meta-data to the tweet text features without any pre-processing. Moreover, we also consider a simpler meta-data approach.

First, we compare *CF* with a simpler combined approach with DL classifiers (aka simpler approach, see Fig. 10(A)). The simpler combined approach works as the proposed combined approach. The only difference is that, in the simpler approach, instead of doing anu pre-processing, the meta-data classifier directly uses meta-data features into the concatenation and classification steps.

Furthermore, we compare *CF* with the simple meta-data approach (aka meta-data approach, see Fig. 10(B)). In the simple meta-data approach, we only use the meta-data classifier without considering the tweet text-based classifier. The results of the experiments are shown in Fig. 11.

From Fig. 11, it can be observed that *CF* outperforms simpler alternatives in terms of accuracy, precision, recall, and f1-score. Over Dataset I, it can be seen that the f1-score of *CF* is 99.68%, whereas, the simpler approach gives the highest 87.56% f1-score. Similarly, in terms of accuracy, precision, and recall, the proposed combined classifier gives better results. Likewise, over Dataset II, it can be observed that the f1-score of the proposed approach is 93.12%, which is greater than the simpler approaches.

Fig. 11 also shows that *CF* outperforms the meta-data approach. Over Dataset I, the proposed combined classifier gives better re-

Table 3
Performance comparison of existing ML-based and DL-based approaches with our proposal.

		Dataset I								Dataset II						
		ML-based				DL-based				ML-based			DL-based			
		A	B	C	D	F	G	Our Approach		A	B	E	F	G	Our Approach	
								Tweet Text	Tweet Text + Meta-data						Tweet Text	Tweet Text + Meta-data
Accuracy	DT	89.92%	87.19%	83.17%	85.45%	57.90%	91.58%	**76.68%**	**99.32%**	91.4%	88.44%	81.89%	65.49%	92%	**61.76%**	**93.38%**
	RF	**93.25%**	**91.38%**	89.55%	94.67%	61.32%	91.1%			**92.1%**	**90.66%**	**85.08%**	67.25%	92.08%		
	LR	90.24%	90.16%	**91.86%**	**99.02%**	65.70%	**91.75%**			55.1%	77.79%	75.32%	68.41%	89.24%		
	SVM	66.43%	72.73%	78.50%	66.3%	**66.43%**	91.54%			74.14%	77.52%	74.70%	**73.73%**	**92.53%**		
Precision	DT	86%	86%	90%	94%	55%	92.74%	**86.5%**	**99.47%**	89%	89%	84%	68%	88.68%	**67.41%**	**95.31%**
	RF	**92%**	**91%**	**92%**	97%	**62%**	93.37%			**91%**	**93%**	**86%**	56%	90.56%		
	LR	88%	88%	91%	**99%**	32%	94.07%			68%	78%	75%	56%	**92.13%**		
	SVM	82%	79%	77%	32%	32%	**95%**			77%	74%	73%	**82%**	90%		
Recall	DT	85%	85%	92%	94%	55%	94.73%	**73.82%**	**99.9%**	89%	89%	85%	68%	**97.31%**	**43.28%**	**91.04%**
	RF	**91%**	**90%**	**93%**	98%	**62%**	93.19%			**92%**	**93%**	**86%**	56%	94.96%		
	LR	88%	89%	90%	**99%**	50%	**93.44%**			67%	78%	72%	53%	90.43%		
	SVM	50%	58%	73%	50%	50%	92.09%			77%	74%	71%	**72%**	96.64%		
F1-score	DT	86%	85%	91%	94%	55%	93.72%	**79.65%**	**99.68%**	89%	89%	84%	68%	92.79%	**50.36%**	**93.12%**
	RF	**91%**	**91%**	**93%**	97%	**62%**	93.27%			**91%**	**93%**	**86%**	55%	92.70%		
	LR	88%	88%	91%	**99%**	39%	**93.75%**			65%	78%	70%	47%	91.27%		
	SVM	40%	55%	74%	39%	39%	93.52%			77%	74%	69%	**70%**	**93.2%**		

sults than the meta-data approach, in terms of accuracy, precision, recall, and f1-score, whereas, over Dataset II, the meta-data approach gives slightly good results only in recall value. On Dataset II, recall value of the proposed approach is 91.04%, whereas, the recall value of the meta-data approach is 94.02%. However, in our application, we consider that precision is more important than recall. Therefore, according to the performance comparison discussed above, we believe that the result confirms the utility of our proposal.

6. Conclusion

In this paper, we have designed a novel deep learning based Twitter spam detection approach to overcome the current issues of existing deep learning and machine learning based spam detection methods. We developed both a text-based classifier, which considers only users' tweet text, and a combined classifier, which considers users' tweet text and meta-data. In the experiments, we showed that our approach gives better results than other existing DL-based and ML-based approaches. Our approach achieves the highest accuracy of 99.68% and 93.12% for dataset I and II, respectively.

In the future, we plan to build a model which can easily classify various types of spammers within different types of social networks, such as Facebook and LinkedIn.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work has received funding from CONCORDIA, the Cybersecurity Competence Network supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 830927.

References

[1] C. Yang, R. Harkreader, G. Gu, Empirical evaluation and new design for fighting evolving twitter spammers, IEEE Trans. Inf. Forensics Secur. 8 (8) (2013) 1280–1293.
[2] M. Fazil, M. Abulaish, A hybrid approach for detecting automated spammers in twitter, IEEE Trans. Inf. Forensics Secur. 13 (11) (2018) 2707–2719.
[3] Twitter Usage Statistics - Internet Live Stats (2019), accessed on 20 April 2019. [Online]. Available: http://www.internetlivestats.com/twitter-statistics/.
[4] F. Benevenuto, G. Magno, T. Rodrigues, V. Almeida, Detecting spammers on twitter, in: Collaboration, electronic messaging, anti-abuse and spam conference (CEAS), 6, 2010, p. 12.
[5] C. Grier, K. Thomas, V. Paxson, M. Zhang, @ spam: the underground on 140 characters or less, in: Proceedings of the 17th ACM conference on Computer and communications security, ACM, 2010, pp. 27–37.
[6] Twitter privacy rules, accessed on 20 April 2019. [Online]. Available: https://help.twitter.com/en/rules-andpolicies/twitter-rules.
[7] A.-Z. Ala'M, J. Alqatawna, H. Faris, Spam profile detection in social networks based on public features, in: 2017 8th International Conference on information and Communication Systems (ICICS), IEEE, 2017, pp. 130–135.
[8] A.K. Ameen, B. Kaya, Detecting spammers in twitter network, Int. J. Appl. Math. Electron. Comput. 5 (4) (2017) 71–75.
[9] C. Chen, J. Zhang, X. Chen, Y. Xiang, W. Zhou, 6 million spam tweets: A large ground truth for timely twitter spam detection, in: 2015 IEEE international conference on communications (ICC), IEEE, 2015, pp. 7065–7070.
[10] K. Lee, J. Caverlee, S. Webb, Uncovering social spammers: social honeypots+ machine learning, in: Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval, ACM, 2010, pp. 435–442.
[11] G. Stringhini, C. Kruegel, G. Vigna, Detecting spammers on social networks, in: Proceedings of the 26th annual computer security applications conference, ACM, 2010, pp. 1–9.
[12] A. Chakraborty, J. Sundi, S. Satapathy, et al., Spam: a framework for social profile abuse monitoring, CSE508 report, Stony Brook University, Stony Brook, NY (2012).
[13] H. Gao, Y. Chen, K. Lee, D. Palsetia, A.N. Choudhary, Towards online spam filtering in social networks., in: NDSS, 12, 2012, pp. 1–16.
[14] M. Mccord, M. Chuah, Spam detection on twitter using traditional classifiers, in: international conference on Autonomic and trusted computing, Springer, 2011, pp. 175–186.
[15] A.H. Wang, Don't follow me: Spam detection in twitter, in: 2010 international conference on security and cryptography (SECRYPT), IEEE, 2010, pp. 1–10.
[16] T. Wu, S. Liu, J. Zhang, Y. Xiang, Twitter spam detection based on deep learning, in: Proceedings of the australasian computer science week multiconference, ACM, 2017, p. 3.
[17] X. Ban, C. Chen, S. Liu, Y. Wang, J. Zhang, Deep-learnt features for twitter spam detection, in: 2018 International Symposium on Security and Privacy in Social Networks and Big Data (SocialSec), IEEE, 2018, pp. 208–212.
[18] A.H. Wang, Detecting spam bots in online social networking sites: a machine learning approach, in: IFIP Annual Conference on Data and Applications Security and Privacy, Springer, 2010, pp. 335–342.
[19] K. Lee, B.D. Eoff, J. Caverlee, Seven months with the devils: A long-term study of content polluters on twitter, in: Fifth International AAAI Conference on Weblogs and Social Media, 2011.
[20] M.M. Swe, N.N. Myo, Fake accounts detection on twitter using blacklist, in: 2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS), IEEE, 2018, pp. 562–566.
[21] I. Idris, A. Selamat, N.T. Nguyen, S. Omatu, O. Krejcar, K. Kuca, M. Penhaker, A combined negative selection algorithm–particle swarm optimization for an email spam detection system, Eng. Appl. Artif. Intell. 39 (2015) 33–44.
[22] L. Araujo, J. Martinez-Romo, Web spam detection: new classification features based on qualified link analysis and language models, IEEE Trans. Inf. Forensics Secur. 5 (3) (2010) 581–590.
[23] S. Lai, L. Xu, K. Liu, J. Zhao, Recurrent convolutional neural networks for text classification, Twenty-ninth AAAI conference on artificial intelligence, 2015.
[24] L. Mou, H. Peng, G. Li, Y. Xu, L. Zhang, and Z. Jin, "Discriminative neural sentence modeling by treebased convolution," arXiv preprint arXiv:1504.01106, 2015 Apr 5.
[25] K.S. Tai, R. Socher, and C.D. Manning, "Improved semantic representations from tree structured long short-term memory networks," arXiv preprint arXiv:1503.00075, 2015 Feb 28.
[26] Convolutional networks, Accessed on 12 June 2019. [Online]. Available: http://cs231n.github.io/convolutional-networks.
[27] Neural networks, Accessed on 12 June 2019. [Online]. Available: https://medium.com/datamonsters/artificial-neural-networks-for-natural-language-processing-part-1-64ca9ebfa3b2.
[28] Y. Kim, "Convolutional neural networks for sentence classification," arXiv preprint arXiv:1408.5882, 2014 Aug 25.
[29] Y. Zhang and B. Wallace, "A sensitivity analysis of (and practitioners'guide to) convolutional neural networks for sentence classification," arXiv preprint arXiv:1510.03820, 2015 Oct 13.
[30] B. Gambäck, U.K. Sikdar, Using convolutional neural networks to classify hate-speech, in: Proceedings of the first workshop on abusive language online, 2017, pp. 85–90.
[31] Y. Kim, Convolutional neural networks for sentence classification. (2017), in: arXiv preprint arXiv:1408.5882.
[32] A.M. Founta, D. Chatzakou, N. Kourtellis, J. Blackburn, A. Vakali, I. Leontiadis, A unified deep learning architecture for abuse detection, in: Proceedings of the 10th ACM Conference on Web Science, 2019, pp. 105–114.
[33] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, 2013 arXiv preprint arXiv:1301.3781.
[34] X. Rong, word2vec parameter learning explained. (2014), in: arXiv preprint arXiv:1411.2738.
[35] D.P. Montminy, R.O. Baldwin, M.A. Temple, E.D. Laspe, Improving cross-device attacks using zero-mean unit-variance normalization, J. Cryptogr. Eng. 3 (2) (2013) 99–110.
[36] Functional-API, Accessed on 12 June 2019. [Online] Available: https://keras.io/guides/functional_api/.
[37] J.B. Diederik P Kingma, Adam: A method for stochastic optimization. (2014), in: arXiv preprint arXiv:1412.6980.
[38] M. Andrychowicz, M. Denil, S. Gomez, M.W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, N. De Freitas, Learning to learn by gradient descent by gradient descent, in: Advances in neural information processing systems, 2016, pp. 3981–3989.
[39] A. Graves, "Generating sequences with recurrent neural networks," arXiv preprint arXiv:1308.0850, 2013 Aug 4.