This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2710540, IEEE Access

Statistical Twitter Spam Detection Demystified: Performance, Stability and Scalability

Guanjun Lin, Nan Sun, Surya Nepal, Jun Zhang, *Member, IEEE,* Yang Xiang, *Senior Member, IEEE,* and Houcine Hassan, *Member, IEEE,*

Abstract—With the trend that the Internet becoming more accessible and our devices being more mobile, people are spending an increasing amount of time on social networks. However, due to the popularity of online social networks, cyber criminals are spamming on these platforms for potential victims. The spams lure users to external phishing sites or malware downloads, which has become a huge issue for online safety and undermined user experience. Nevertheless, the current solutions fail to detect Twitter spams precisely and effectively. In this paper, we compared the performance of a wide range of mainstream machine learning algorithms, aiming to identify the ones offering satisfactory detection performance and stability based on a large amount of ground truth data. With the goal of achieving real-time Twitter spam detection capability, we further evaluated the algorithms in terms of the scalability. The performance study evaluates the detection accuracy, the TPR/FPR and the F-measure; the stability examines how stable the algorithms perform using randomly selected training samples of different sizes. The scalability aims to better understand the impact of the parallel computing environment on the reduction of training/testing time of machine learning algorithms.

Index Terms—Machine learning, Twitter, spam detection, parallel computing, scalability

1 Introduction

Online social networks (OSNs), such as Twitter, Facebook and LinkedIn, have had significant impact on our life and have reshaped the way we socialize and communicate. Thanks to the OSNs, we are able to get in touch with our friends and family anywhere, anytime. Take Twitter for example, we are able to post messages with pictures, videos, text and follow others whom we are interested in and care for. So far, Twitter has gained tremendous popularity and have had up to 313 million active users [24].

However, with the increasing number of users on Twitter, the spamming activities are growing as well. Twitter spams usually refer to tweets containing advertisements, drugs sales or messages redirecting users to external malicious links including phishing or malware downloads, etc. [1]. Spams on Twitter not only affect the online social experience, but also threatens the safety of cyberspace. For example, in September 2014, New Zealand's network was melt down due to a malware downloading spam [18], the result of which signaled the alarm of Twitter spams. Therefore, there is an urgent need to effectively combat the spread of spams. Twitter has applied rules to regulate users' behaviors such as restricting users from sending duplicate contents, from mentioning other users repeatedly or from posting URL-only contents [5]. Meanwhile, the spamming issues have attracted the attention of the research community. Researchers have put many efforts to improve Twitter spam detection efficiency and accuracy by proposing various novel approaches [1],[23],[25],[32],[28].

To the best of our knowledge, there are three major types of feature-related solutions for Twitter spam detection. The first type is based on the features of user account and tweets content, such as account age, the number of followers/followings and the number of URLs contained in the tweet, etc. These features can be directly extracted from tweets with little or without computation. Based on the observed facts that the content-based features could be fabricated easily. Other researchers proposed to use robust features derived from the social graph [31], which is the second type of solution. Yang et al. [31] proposed a spam detection mechanism based on the graph-based features, known as Local Clustering Co-efficiency and Betweenness Centrality. In [23], the authors present a directed graph model to explore the relationship of senders and receivers. Nevertheless, graph-based features are empirically difficult to collect, because generating a large social/relationship graph can be time and resource consuming considering that a user may interact with a large but unpredictable number of users. The third type of solution focuses on tweets with URLs. According to [34], domains and IP blacklists are used to filter tweets containing malicious URLs. Both [26] and [29] applied the URLs based features for Twitter spam detection.

However, there is a lack of comparative work benchmarking the performance of machine learning algorithms on Twitter spam detection to demonstrate the applicability and feasibility of these machine learning approaches in the real-world scenarios. In this paper, we bridge the gap by conducting an empirical study of 9 commonly-used machine learning algorithms to evaluate the detection performance in terms of detection accuracy, the true/false positive rate (TPR/FPR) and the f-measure, as well as the detection

- *Guanjun Lin, Nan Sun and Jun Zhang are with the School of Information Technology, Deakin University, Geelong, Victoria 3216, Australia*
- *Jun Zhang is the corresponding author (Email: jun.zhang@deakin.edu.au).*
- *Surya Nepal is with Data61, CSIRO Melbourne, Victoria 3008, Australia.*
- *Yang Xiang is with the School of Information Technology, Deakin University, Melbourne, Victoria 3125, Australia*
- *Houcine Hassan is with the Department of Computer Engineering in the Universitat Politcnica de Valncia, Spain. Camino de Vera, s/n 46022, Valencia Valencia, Espaa*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2710540, IEEE Access

stability based on 200k training samples[1]. We simulated the realistic-like condition by using the unbalanced ratio of spam and non-spam datasets for performance evaluation of the selected algorithms. Additionally, we study the scalability of different algorithms to examine the efficiency of running on parallel computing environment. Based on the experiments, we explore and discuss the feasibility of achieving real-time detection capability considering various applicable scenarios[2]. In summary, the contributions of this paper are the following:

- The detection performance of 9 mainstream algorithms are compared for identifying the best-performed algorithms on our light-weight feature datasets. We found that two decision tree based algorithms – Random Forest and C5.0 achieved the best performance in various conditions.
- The stability of each algorithm was studied by repeating the experiments with different sizes of datasets to examine the performance fluctuation, aiming to find a robust algorithm in terms of performance. Our experiment demonstrated that Boosted Logistic Regression (BLR) achieved a relatively stable detection performance regardless of the size of training samples, and tree-based algorithms such as random forest and C5.0 performed stably under the situation where training samples are randomly selected.
- The scalability of selected algorithms is analyzed by measuring how the algorithms' training time varies when the number of hosting CPU cores doubled (1, 2, 4, 8, 16 and 32) each time. Empirical results show that deep learning scaled up the best, capable of achieving real-time detection giving sufficient computational resources.
- A superlinear speedup was achieved by deep learning on our datasets. The last subsection of section VII is an attempt to explain this finding.

The rest of this paper is organized as follows. The related work is presented in section II. Section III describes the algorithms selection procedure. In section IV, we address our data collection procedure and the light-weight feature extraction approach. Section V introduces how the experiments are setup and evaluated. Then, the results and findings are presented in Section VI, followed by the analysis and discussion of the experiment outcomes and the feasibility of real-time spam detection. The conclusions and future work are in section VIII.

2 Related Work

Twitter spam detection is an important topic in social network security. Many pioneer researchers have been devoted to the study of spam detection on the OSNs. However, due to the constant adaptation to the spam detection techniques, spammers are still active on the OSNs. Hence, to combat the spread of spams, a series of methods and solutions have been proposed based on different types of features. Some works relied on the user profile features and message content features to identify spams; some proposed using graph-based features, typically the distance and connectivity of a social graph; and some others relied on embedded URLs as the means of spam detection features.

The user profile and message content based features can be extracted with little computation, thus, it is practical to collect a large amount of account information and sample messages for analysis and research. In [5], the authors used the account and content features, such as the time period an account has existed, the number of follower and following, the number of hashtags and URLs embedded in the message to detect the spam tweets. Apart from considering the account and content features, [1] and [25] also takes the user behavior features into account. In [1], more detailed behavior-related features were considered, such as existence of spam words on the users nick name, the posting frequency and the number of tweets posted per day and per week.

Although the profile and content features could be collected conveniently, it is possible to fabricate and modify these features to escape detection [23]. Hence, some improvement attempts to advocate using graph-based features for identifying spams. In [23], Song et al. proposed a novel spam filtering system, relying on the sender-receiver relationships. Based on the analysis of the distance and connectivity of the relation graph, the system predicts whether an account is spammer or not. Their method achieved a very high accuracy but was not applicable in real-time detection for its unsatisfactory computational performance and some unrealistic assumptions. Yang et al. [32] designed some robust features such as graph and neighbor-based features to cope with the spammers who are constantly evolving their techniques to evade the detection. They proved that their approach achieved higher detection rate and lower false positive rate compared with some previous works. In [28], the authors combined graph-based and content-based features to facilitate the spam filtering.

Although detecting twitter spam using social graph features could achieve decent performance, collecting these features can be time-consuming because the Twitter user graph is huge and complex. Due to not being practical to apply social graph based features to a real-time spam detection system, there are some works using embedded URLs in tweets for spam detection on Twitter under the assumption that all spam tweets contain URLs. In [26], Thomas et al. developed the Monarch which is a real-time system for detecting spams by crawling URLs. They used features extracted from URLs, for example, the Domain token, the path tokens and the URLs query parameters as detection criteria. Moreover, in [29], Wang et al. focused on the detection of spam short URLs. They collected both the short URLs and the click traffic data for characterization of the short URL spammers. Based on the click traffic data, they classified the tweets with URLs into spams and non-spams, and achieved more than 90% of accuracy.

1. Part of this dataset is publicly available in our website: http://nsclab.org/nsclab/resources/. For a complete dataset, please contact us.

2. A real-time Twitter spam detection prototype system was implemented based on the experiments, the source code can be found at: https://github.com/danielLin1986d/RTTSD_prototype.git

TABLE 1: 5 Categories of The Chosen Algorithms

Categories	Algorithms
kNN-based	kNN
	Weighted kNN (k-knn)
Decision tree-based	Random Forest
	C5.0
Boosting Algorithms	Stochastic Gradient Boosting Machine(GBM)
	Boosted Logistic Regression (BLR)
Bayesian Algorithms	Naive Bayes
Neural Network-based	Neural Network
	Deep Learning

3 Algorithms For Spam Detection

Machine learning techniques, which offer computers the capability of learning by extracting or filtering useful information or patterns from raw data, have been widely applied in diverse fields [6]. A variety of different machine learning algorithms have been developed and improved for targeting diverse application scenarios and data types. In this paper, we choose 9 supervised machine learning algorithms for spam/non-spam tweets classification. The selected algorithms can be categorized into 5 groups as shown in Table 1.

The reasons that we choose these algorithms are as follows:

- The selected algorithms are widely used both in industrial and academic fields. We would like to investigate how well these algorithms perform with respect to our tweets' datasets on various conditions.
- To achieve real-time detection, we used only 13 features. Therefore, kNN-based algorithms was chosen due to their suitability for data samples with relatively small number of dimensions [10]. Apart from kNN, we also selected k-kNN [12] which is an improved kNN algorithm. It introduces a weighting mechanism for measuring the nearest neighbors based on their similarity to an unclassified sample. So the probability of a newly observed sample belonging to a class is influenced or weighted by its similarity to the samples in the training set.
- Decision tree-based and boosting algorithms are two main categories of popular machine learning tools which have gained tremendous attentions in data mining and statistical fields. Among these algorithms, random forest [2] and C5.0 [19] are selected as the representatives of the decision tree-based algorithms, and stochastic gradient boosting machine (GBM) [9] and Boosted Logistic Regression (BLR) [8] are chosen to represent the boosting family.
- Among the Bayesian algorithms, the Nave Bayes, being a classic probabilistic classifier which builds on the assumption that all features of data are probabilistically independent [3], is selected as a candidate for our experiments.
- The neural network and deep learning [15] (implemented using the deep neural networks architecture) are chosen in contrast to each other. At the time of writing, deep learning is the most popular machine learning algorithm which has achieved practical success in the fields of computer vision, speech recognition and natural language processing [16].

4 Data Collection and Feature Selection

4.1 Data Collection

4.1.1 Collection Procedure

As the continuation and extension of our work [5], we used the same data set as the one described in the previous work. Totally, we collected 600 million tweets, all of which contain URLs. Based on [7] and [35] the majority of spam tweets embedded URLs to attract victims with malicious purposes. Therefore, our research builds on the assumption that all spam tweets contain URLs with the purpose of luring users to external phishing sites or malware downloading.

4.1.2 Ground Truth

With the help of Trend Micros Web Reputation Service[17], we were able to identify 6.5 million spams tweets whose URLs are identified as malicious. In some works, researchers had to label spam tweets manually based on Blacklisting service, such as Google SafeBrowsing. Thanks to Trend Micros WRS, it could automatically check whether the URLs embedded in tweets are malicious by using its large dataset of URL reputation records. We define tweets with malicious URLs as spams. Due to the Trend Micros WRS devoting to collecting latest URLs and maintaining the database of URLs, the URLs reputation list is up-to-date, which ensures the accuracy of labelling. We identified 6.5 million spam from 600 million tweets, and we have obtained up to 30 million labelled tweets data in total to form our ground truth dataset. Our experiment data was randomly selected from this labelled dataset.

4.2 Feature Selection

4.2.1 Light-weight Statistical Features

It is a non-trivial task to select and decide how many features are needed for training a model that guarantees a satisfying prediction capability while maintaining acceptable size of feature set to ensure prediction efficiency. Namely, it is important to trade off between the number of features and the predicting power of the trained model. In this paper, we have chosen 13 features from our randomly selected ground truth data as summarized in Table 2. We applied the same approach as described in [5] for selecting and extracting features from our ground truth dataset, because these 13 features can be easily extracted from the tweets collected through Twitter's Public Streaming API. Hence, little computation effort was required. Besides, using small size features for spam detection helps to reduce the computational complexity during the model training and testing processes, which makes real-time spam detection achievable even with massive training/testing data.

The extracted features can be categorized into two groups: user profile-based features and tweet content-based features. The user profile-based features include: the time period of the account has existed, the number of followers,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2710540, IEEE Access

the number of following, the number of favorites this user received, the number of lists user is a member of and the number of tweets this user sent. These 6 features depict the behaviors of the user account. The tweet content-based features include: the number of this tweet has been retweeted, the number of favorites this tweet received, the number of hashtags, the number of times this tweet being mentioned, the number of URLs included, the number of characters and the number of digits in this tweet. We believe that if an account is controlled by a spammer, the tweets sent by this account would be spams. However, if an account initially belonged to a legitimate user and then was comprised by a spammer, the account could send out normal tweets and spams. Therefore, it is necessary to analyze both the user behaviors features and the tweet content features for deciding spams and non-spams.

5 Environment and Evaluation Metrics

5.1 Experiment Environment and Experiment Setup

5.1.1 Hardware Specification

We conducted our experiments on a workstation equipped with two Physical Intel(R) Xeon(R) CPU E5-2690 v3 2.60GHz CPUs, totally providing 48 logical CPU cores, capable of offering a small scale of parallelism. The workstation offers 64 GB memory and 3.5 TB storage which satisfies our required experiment conditions.

5.1.2 Software Tools and Platform

Our software environment was built on Ubuntu Server 16.04 LTS. We used R programming language (version 3.2.3) with caret package (version 6.0-68) [13] and doParallel package (version 1.0.10) [30] to perform our experiments. The caret package encapsulates a variety of machine learning algorithms and tools. It acts like an API, allowing users to invoke different algorithms and tools using a standardized syntax and format [14]. To call a certain algorithm for training, a user needs to specify "method" parameter when invoking the train function. In our experiment, we called eight selected algorithms provided by "caret" package[3]. Table 3 shows the "method" parameter we used for each algorithm we called in our experiment.

The default resampling scheme provided by the train function of the "caret" package is bootstrap [14]. In our experiment, we used 10-fold cross-validation by specifying "repeatedcv" parameter equals 1 (performing 10-fold cross-validation 1 time) during the classifier training process.

By default, all selected algorithms invoked by the "caret" package can only utilize one CPU core. Therefore, the doParallel package was used to enable the algorithms to be executed in parallel to take advantage of multiple CPU cores [30]. Specifically, the doParallel package acts as a "parallel backend" to execute the *foreach* loops in parallel. It also allows users to specify how many CPU cores can be used simultaneously.

We used a powerful big data analytic tool called H2O (version 3.8.3.3) [11] to implement deep learning algorithm using the architecture of deep neural networks [4]. The H2O is an open-source and web-based platform which allows users to analyze big data with different built-in algorithms. It also offers flexible deployment options enabling users to specify the usage of computational resources such as the number of CPU cores, the amount of memory, and the number of computing nodes.

3. We called the algorithms using the default settings without specifically tuning the parameters.

5.2 Evaluation Metrics

This section addresses the metrics chosen for evaluating the performance, stability and scalability of the selected 9 algorithms.

5.2.1 Performance

The measure of performance is to use the accuracy, the true positive rate (TPR), the false positive rate (FPR) and the F-measure as metrics. The accuracy is the percentage of correctly identified cases (both spams and non-spams) in the total number of examined cases, which can be calculated using equation (1) . The TPR a.k.a recall, indicates the ratio of correctly identified spams to the total number of actual spams. It can be calculated using equation (2). The FPR refers to the proportion of non-spams incorrectly classified as spams in the total number of actual non-spams, as equation (3) shows. The precision is defined as the ratio of correctly classified spams to the total number of tweets that are classified as spams, as shown by the equation (4). Lastly, the F-measure a.k.a F1 score or F-score, is another measurement of prediction accuracy combining both the precision and recall. It can be calculated by the equation (5).

$$Accuracy = TP + TN/(TP + TN + FP + FN) \quad (1)$$

$$TPR = TP/(TP + FN) \quad (2)$$

$$FPR = FP/(FP + TN) \quad (3)$$

$$Precision = TP/(TP + FP) \quad (4)$$

$$F - measure = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \quad (5)$$

With the same hardware configuration, we evaluate the performance of each algorithm under various experiment conditions using the above-mentioned metrics. Firstly, we used different sizes of the training data ranging from 2k to 200k as shown in the dataset 1,2 and 3 in TABLE 4. But for both training and testing data, we kept the ratio of spams and non-spams to be 1:1. For the amount of testing data, we used 100k spam and 100k non-spam (totally 200k) tweets to test the performance of trained classifiers.

In reality, the number of spam and non-spam is not evenly distributed. The spams only account for less than 5% of the total number of tweets. In order to simulate the real scenario, the ratio of non-spam to spam of the testing data is set to 19:1 (as shown in the datasets 4, 5 and 6 in Table 4). But we kept the ratio of training data unchanged and still used 2k, 20k, 200k datasets to train all the classifiers.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2710540, IEEE Access

TABLE 2: 13 Extracted Features and Feature Descriptions

Feature Category	Feature Name	Description
Account-based features	account_age	The age of an account
	no_follower	The number of followers
	no_following	The number of followings
	no_userfavorites	The number of favourites this user received
	no_lists	The number of lists the user is a member of
	no_tweets	The number of a user posted tweets
Tweet content-based features	no_retweets	The number of times this tweet has been retweeted
	no_tweetfavorites	The number of favourites this tweet received
	no_hashtag	The number of hashtags in this tweet
	no_usermention	The number of times this tweet being mentioned
	no_urls	The number of URLs contained in this tweet
	no_char	The number of characters in this tweet
	no_digits	The number of digits in this tweet

TABLE 3: The "method" Parameter Specified Using "caret" Package in Experiments

The Selected Algorithms	The "method" Parameter Specified
K Nearest Neighbor	knn
Weighted K Nearest Neighbours	kknn
Naive Bayes	nb
Random Forest	rf
C5.0	c5.0
Boosted Logistic Regression	LogitBoost
Stochastic Gradient Boosting Machine	gbm
Neural Network	pcaNNet

5.2.2 Stability

The stability measures how stable each algorithm performs in terms of detection accuracy. We apply the standard deviation (SD) to quantify stability. When measuring the performance for each algorithm, we repeated 10 times to examine how the detection accuracy varies each time due to the random selection of the training samples. Then, we studied how different sizes of training data affects the accuracy of each algorithm. By recording the accuracy of each algorithm for 10 times on various sizes of data, we calculated the standard deviation value for every algorithm. A large SD value implies the fluctuation in detection accuracy and instability of the performance.

5.2.3 Scalability

The scalability is used to evaluate how the algorithms scale on the parallel environment which is a shared-memory multi-processor workstation. The parallel implementation enables algorithms to make full use of multiple CPUs to accelerate the calculation process by executing tasks on these CPUs simultaneously. To measure scalability[4], we use the term speedup to quantify how much performance gain can be achieved by a parallel algorithm compared with its sequential counterpart [36] or compared with the parallel algorithm using one process. There is a formula for calculating the speedup:

$$S(p) = T_1/T_p$$

where $S(p)$ is the speedup value, and p denotes the number of processors used; T_1 refers to the execution time needed to run the parallel algorithm using a single processor; T_p is the execution time needed to run the parallel algorithm using p processors with the same problem size [27]. The analysis of speedup trends helps to understand the relationship between the performance gain (reflected as the decrease of execution time) and the amount of computational resources involved.

4. In this paper, we narrow down the discussion of scalability to the fix-size problems.

6 Results and Analysis

6.1 Performance

6.1.1 Accuracy on Evenly Distributed Datasets

As shown in Fig.1, generally, a larger amount of training data contributes to higher classification accuracy under the condition that the testing data is evenly distributed. Specifically, with the size of training data increasing from 2k to 20k, then to 200k, the accuracy of all algorithms raises, except for BLR. Noticeably, C5.0 and random forest achieved more than 90% accuracy when trained with 200k tweets, which is the highest accuracy observed among all. k-kNN gained the third highest accuracy which was around 85% when trained with 200k samples. As for GBM, Naive Bayes, Neural Network and Deep Learning, once the size of training data reaches 20k, there is no substantial increase observed in accuracy. However, for BLR, changing the size of training data exerts no influence on its classification accuracy.

6.1.2 Accuracy on Unevenly Distributed Datasets

When using unevenly distributed datasets with 1:19 spam to non-spam ratio, it is unexpected to see that the majority of the selected algorithms did not experience a substantial decline regarding their detection accuracy, as illustrated in Fig.2. Similarly, C5.0 and random forest gained more than 90% of accuracy, followed by k-kNN which was 85%. However, obvious drops of accuracy were seen on kNN and Naive Bayes.

6.1.3 Performance Comparison between Evenly and Unevenly Distributed Datasets

In this section, the impact of uneven ratio of spam to non-spam on the detection performance is evaluated by comparing the evenly distributed groups with the uneven distributed ones (dataset 1 and 4 compared with dataset 3 and 6, please refer to Table 4).

Fig. 3-(a) is the comparison of the FPR values of the algorithms on Dataset 1 and 4. It shows that except for Naive Bayes, there was no significant change of the FPR

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2710540, IEEE Access

TABLE 4: The Training and Testing Datasets with Different Spam to Non-spam Ratios

	Training Data		Testing Data	
Dataset	NO. of Spam Tweets	NO. of Non-spam Tweets	NO. of Spam Tweets	NO. of Non-spam Tweets
1	1000	1000	100000	100000
2	10000	10000	100000	100000
3	100000	100000	100000	100000
4	1000	1000	10000	190000
5	10000	10000	10000	190000
6	100000	100000	10000	190000

TABLE 5: The Comparison of the Confusion Matrix of k-Nearest Neighbor on Different Spam to Non-spam Ratios

		Dataset 1		Dataset 4	
		NO. of spams	NO. of Non-spam	NO. of Spam	NO. of Non-spam
True Condition	NO. of spams	61042	39020	6279	3724
	NO. of Non-spams	37558	62381	76298	113653
		Predicted Condition			

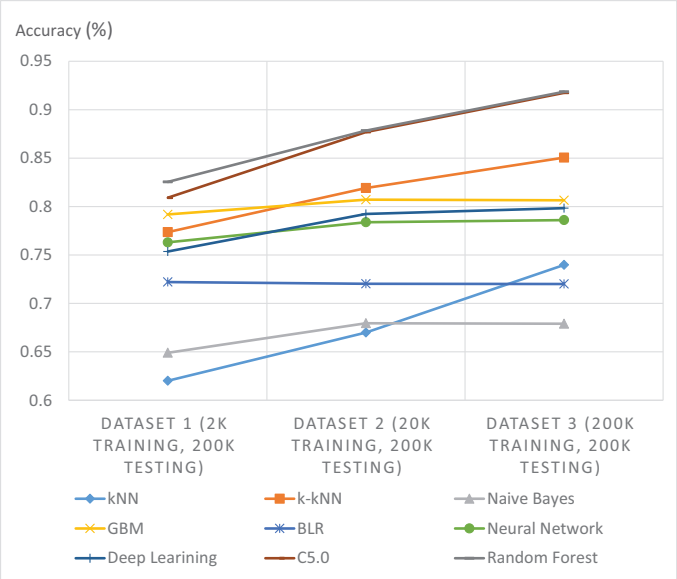


Fig. 1: Detection Accuracy (%) of 9 Algorithms Using Dataset 1, 2 and 3 with the ratio of spams and non-spams being 1:1

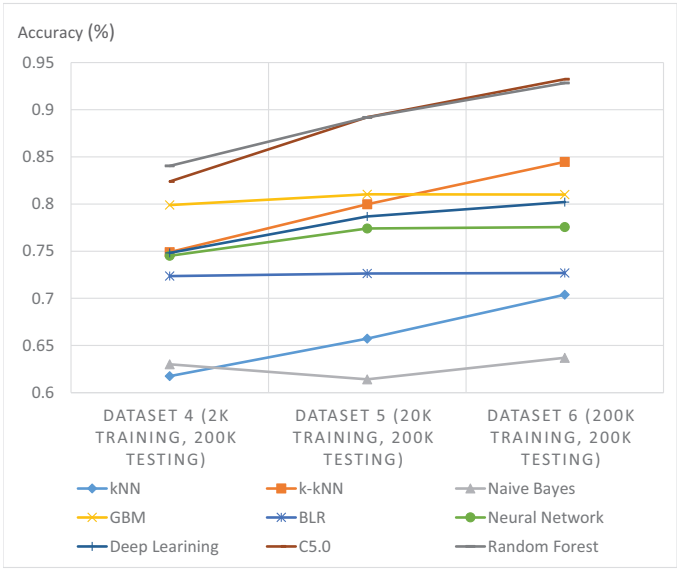


Fig. 2: Detection Accuracy (%) of 9 Algorithms Using Dataset 4, 5 and 6 with the ratio of spams and non-spams being 1:19

values observed when the ratio of spam to non-spam was uneven. For Naive Bayes, its FPR value dropped from 50% to around 37% when using unevenly distributed testing data. These differences seen in the FPR values were caused by the randomly selected training datasets.

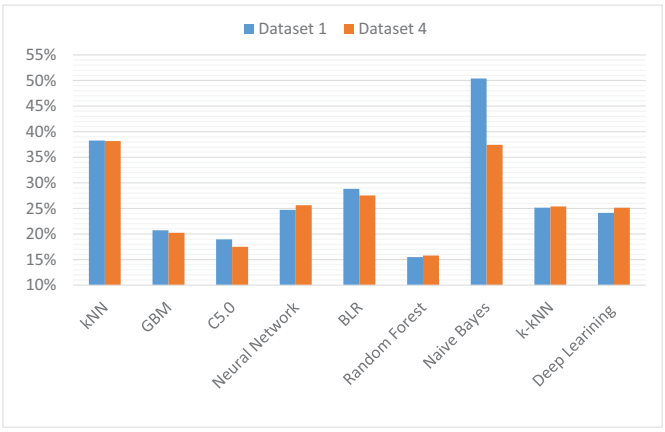
Similarly, Fig. 3-(b) shows that the two groups of TPR values varied slightly, apart from BLR and Naive Bayes. For BLR, only 2% of drop was seen when using the uneven ratio dataset. However, for Naive Bayes, there was approximate 8% of decline seen.

For F-measure values, as illustrated on Fig. 3-(c), when using 2k trained classifiers to run on the unevenly distributed testing dataset, there was an substantial decrease observed for all algorithms. Generally, the F-measure values of all algorithms on Dataset 4 were less than half of the F-measure values on Dataset 1. Especially, the F-measure of kNN was 60% when using the 1:1 spam to non-spam ratio of testing data, but when using the 1:19 ratio testing data, it dropped to around 14%. Likewise, there was an significant decline seen on Naive Bayes. Comparatively, the decrease of the F-measure values of C5.0 and Random Forest was not as dramatical as that of other algorithms. For Random Forest, the F-measure declined to half when using the unevenly distributed testing dataset.

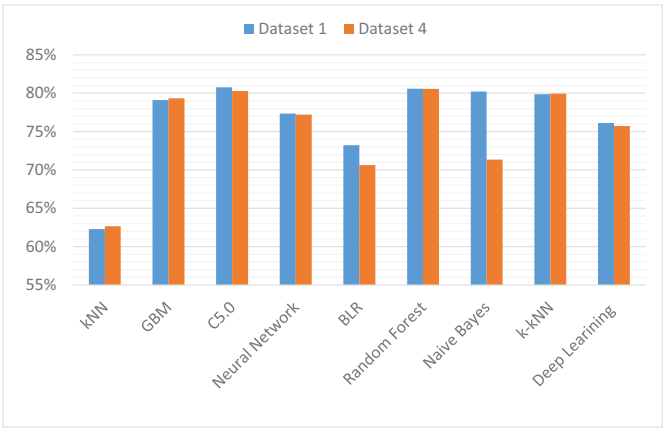
Although increasing the size of the training dataset to 200k contributed to the decrease of the FPR values and the growth of the TPR values as shown in Fig. 4-(a) and Fig. 4-(b), there was still a substantial drop in the F-measure values on the unevenly distributed testing samples. As addressed in Fig. 4-(c), apart from C5.0 and Random Forest, the F-measure values of the algorithms on unevenly distributed dataset dropped to one third of the values on evenly distributed datasets. Particularly, for kNN and Naive Bayes, the F-measure values of both algorithms dropped from around 70% to 20%, which is a 50% of decrease. But for C5.0 and Random Forest, they only experienced approximate decrease of 30%.

To figure out the reason of a substantial decline in the F-measure values, we use the confusion matrix which is one of our tests using k Nearest Neighbor as an example to demonstrate the cause of the drop. Due to the F-measure

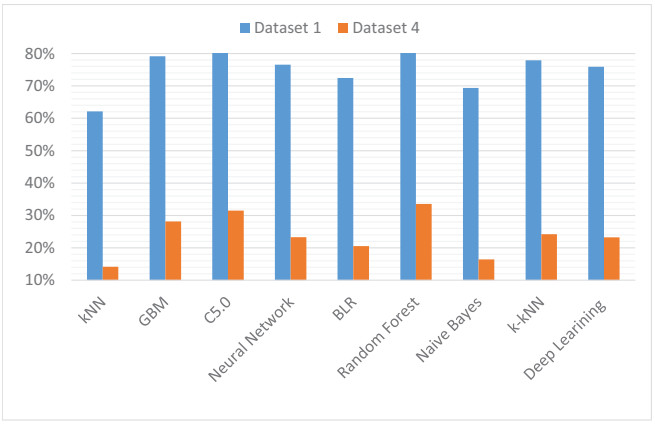
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2710540, IEEE Access



(a) The FPR Values on Dataset 1 VS 4

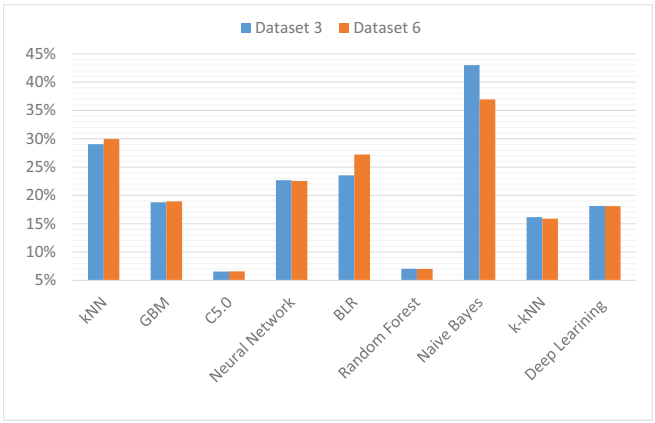


(b) The TPR Values on Dataset 1 VS 4

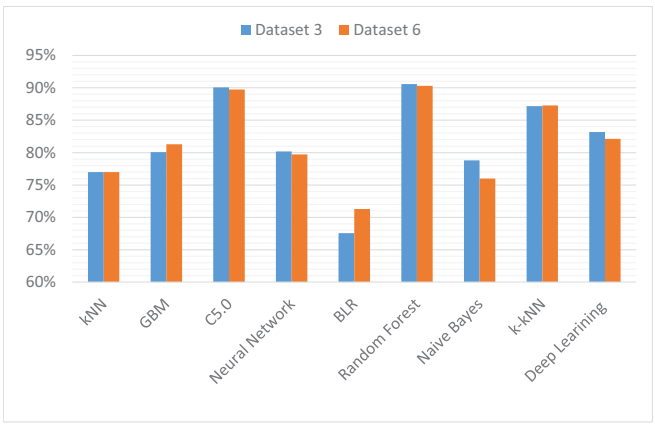


(c) The F-measure Values on Dataset 1 VS 4

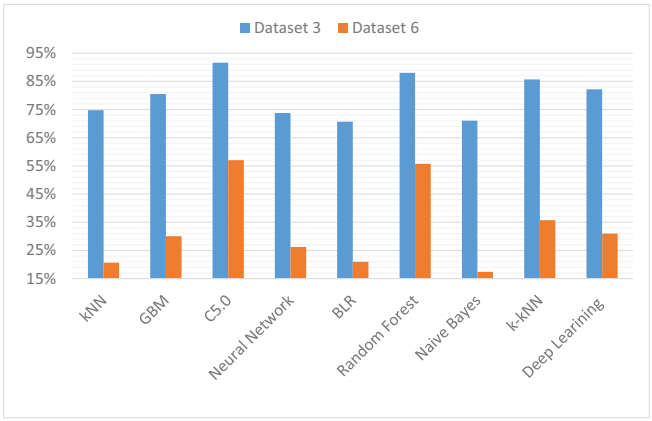
Fig. 3: Performance Comparison of All Algorithms on Dataset 1 VS 4



(a) The FPR Values on Dataset 3 VS 6



(b) The TPR Values on Dataset 3 VS 6



(c) The F-measure Values on Dataset 3 VS 6

Fig. 4: Performance Comparison of All Algorithms on Dataset 3 VS 6

being the combination of recall and precision, we need to evaluate both values.

Firstly, we check the TPR. As the Table 5 shows that for kNN algorithm, when using the 2k training and 200k evenly distributed testing dataset, 61042 spam tweets are correctly classified, while 39020 spams were incorrectly classified as non-spams. Hence, we had 61% of TPR. When using the 2k training and 200k unevenly distributed testing dataset, although 6279 spam tweets had been correctly identified as spams meaning that the algorithm had detected 62% of the spams tweets, there were 3724 spams that had been incorrectly identified as non-spams. So, the TPR was round 62% when using the uneven dataset, and there is only 1% of difference in TPR, which obviously is caused by the randomization of training data selection.

Secondly, as Table 5 illustrated that when the spam to non-spam ratio is equal (using dataset 1), there was 37558 of the non-spams incorrectly classified as spams, resulting in a

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2710540, IEEE Access

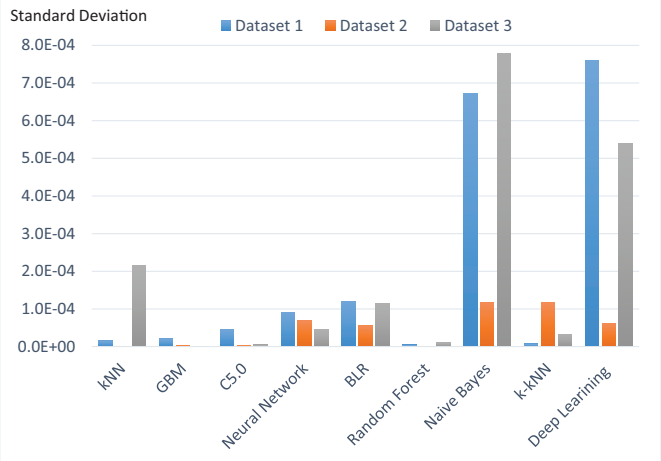


Fig. 5: The Comparison of Accuracy of The Standard Deviation of All Algorithms with Evenly Distributed Dataset 1, 2 and 3

precision of 62%. However, due to the dramatical increase of non-spams, when using the 1:19 spam to non-spam ratio dataset, there was 76298 non-spam incorrectly classified as spams. Hence the precision dropped significantly to 7.6%. Consequently, a significant drop in the precision value causes the substantial decrease in the F-measure when using more non-spams than spams.

Similarly, for other algorithms, due to the involvement of more non-spam tweets in the testing datasets (19 times more non-spams than spams), more number of the non-spam tweets were wrongly put into the spam class, causing the F-measure values to drop.

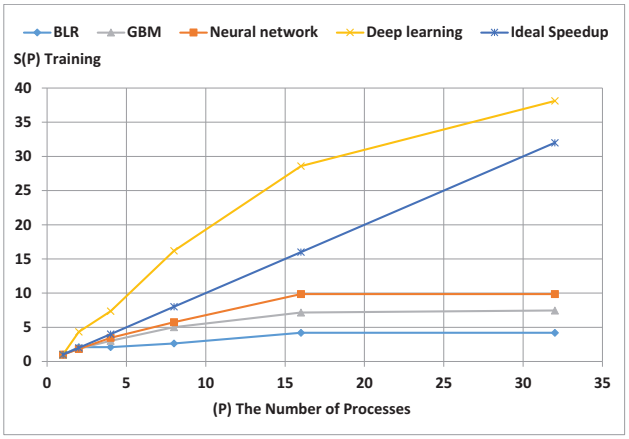
6.1.4 Stability

The stability reveals how the algorithms' detection accuracy are affected by the randomly selected testing and training data, and the variation of the size of training data. The Fig.5 shows a general performance stability distribution of all algorithms on evenly distributed datasets during 10 times repeated experiments. Generally, Random Forest, C5.0 and GBM performed stably across 3 datasets. In contrast, Naive Bayes and Deep Learning performed in an unstable manner, especially when using the 2k and 200k training datasets. Specifically, when trained using 20k training dataset, all algorithms showed a relatively stable performance compared with that on dataset 1 and 3.

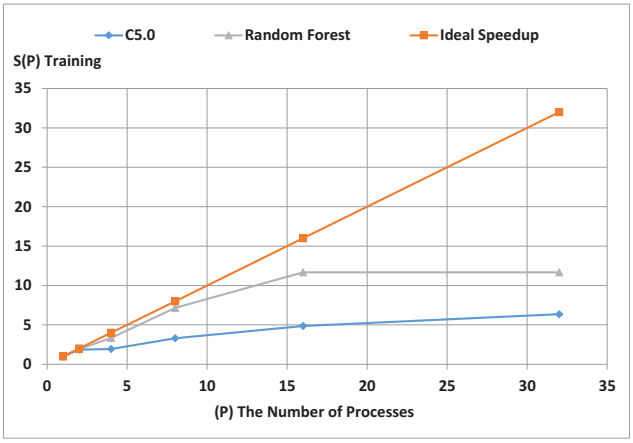
6.1.5 Scalability

Based on our experience, classifiers need to be trained frequently to adjust to the constant changing trend of the tweets data. Therefore, in this paper, we focus on discussion of the training time speedup of the algorithms.

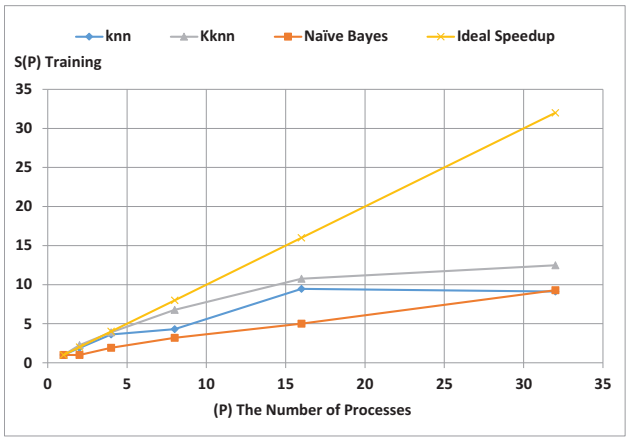
According to the speedup formula introduced in subsection "Evaluation metrics", we define the time spent in training using one CPU core as T_1, and T_p is the time spent in training using p CPU cores with the same data size. Then, we calculate the speedup value $S(p)$. Generally, $S(P) < p$ applies to most of the situations on parallel environment. When $S(P) = p$, it implies that using p CPU cores is exactly p times faster than using 1 CPU core with the problem size unchanged. This is called the *ideal speedup* [33] or *linear speedup* [22]. Whereas, when $S(P) > p$, it is called *superlinear speedup* [22].



(a) Scalability Comparison of BLR, GBM, Neural Network and Deep Learning with Ideal Speedup



(b) Scalability Comparison of C5.0 and Random Forest with Ideal Speedup



(c) Scalability Comparison of kNN, k-kNN and Naive Bayes with Ideal Speedup

Fig. 6: The Scalability Comparisons of All Algorithms with Ideal Speedup as The Benchmark, As The Number of CPU Cores Increases from 1 to 32

In Fig. 6, the speedup trends of all algorithms are

presented in three figures for a rough comparison[5] and we choose the ideal speedup as a reference benchmark to compare with the speedup values of all algorithms.

The Fig.6-(a) compares BLR, GBM, Neural Network and Deep Learning with the ideal speedup. It shows that deep learning achieves a superlinear speedup depicted by the yellow line, while the speedup trends of other algorithms all lie beneath the trend of the ideal speedup. Specifically, with the number of CPU cores increasing from 1 to 16, there is a steady growth in the speedup value of Deep Learning. When the number of CPU cores doubled from 16 to 32, the speedup value increases continuously but the growth rate slows down. Based on the speedup trend, we could anticipate that adding more CPU cores would still be able to substantially reduce the training time of Deep Learning algorithm.

Compared with GBM and BLR, neural network demonstrates a better speedup. As for their speedup trend, one can see that GBM, BLR and neural network increase gradually when the number of CPU cores changing from 1 to 16. However, with more CPU cores added in, no obvious increase is observed, implying that under current problem size, involving more computational resources would not accelerate the training process of these algorithms.

The Fig.6-(b) compares the C5.0 and Random Forest with the ideal speedup. It shows that there is an continuous growth seen on the speedup trends of Random Forest and C5.0 as represented by the gray and blue lines, when the number of CPU cores reaches to 8. As the number hits 16, there is still a modest growth in the speedup trends for Random Forest. But for C5.0, the growth rate of the speedup value is slight. When the number of CPU continues to increase to 32, there is on obvious rise in speedup trends for both algorithms.

The similar speedup trends happen to kNN and k-kNN as well, and there are overlaps between the trends of these two algorithms when using less than 8 CPU cores, as demonstrated in Fig.6-(c). Likewise, Naive Bayes also demonstrated a similar speedup trend. The difference is that when the number of CPU cores doubles from 16 to 32, there was a slight increase seen in the speedup value of Naive Bayes, indicating that there is some rooms for shortening training process of Naive Bayes.

7 Discussion

7.1 Observations of Performance and Stability Evaluation

Based on the experiment outcomes, both Random Forest and C5.0 outperformed other algorithms in terms of detection accuracy, the TPR/FPR and the F-measure when using the evenly distributed datasets. Despite algorithms experienced a significant drop in their F-measure values with the unevenly distributed datasets, Random Forest and C5.0 still achieved relatively high F-measure values, proving that they perform more stably than other algorithms. Hence, in terms of prediction capability, Random Forest and C5.0 are ideal candidates for real world spam detection based on our datasets, and it is reasonable to speculate that the tree-based algorithms could yield moderate performance on similar datasets.

It is worth mentioning that Deep Learning also achieved approximately 80% of accuracy when trained with 200k training data. Its TPR and F-measure values were above or equal to 80% when using the evenly distributed datasets, which makes Deep Learning algorithm stand out.

Intuitively, using more data for training helps to form a classifier offering better performance. This has been proved by our experiments as shown in Figs 1 and 2. However, there are exceptions. For instance, there was no considerable increase in accuracy for GBM, Naive Bayes and Neural network once the size of training data reached 20k. One of the possible reasons could be that when applying these algorithms on our datasets, using 20k training data has provided sufficient information for training a model that clearly depicts the boundaries of spams and non-spams. Therefore adding more training samples would not further benefit the efficacy of classification. It may have the risk of causing over-fitting.

The experiment of stability proved again that Random Forest performed more stably than other algorithms did. This gives an extra point for Random Forest, making it more suitable for the real world spam detection system. Beside, GBM and C5.0 also show stable performance under randomly selection of training samples, compared with other algorithms.

Nevertheless, the performance and stability of the algorithms achieved on our datasets should serve only as a reference. Algorithms may behave differently on different tweets' datasets or using different features. With careful tuning the parameters of selected algorithms, their performance can be further improved.

7.2 Observations of Scalability Evaluation

However, when taking the real world situation into account, both the time an algorithm spent in training and testing should be considered. As it is demonstrated by the experiments that for most of the selected algorithms, the more training data used, the better detection accuracy achieved. Hence, in order to guarantee the utility in real situation, a trade-off between classifiers' prediction/classification accuracy and the amount of training samples should be made.

The training time of all algorithms on 200k datasets using 32 CPU cores is described in Table. 6. One can see that it took the C5.0 more than 2 hours to complete the training, while BLR only needed 3 seconds for the same problem size. Although Random Forest has achieved more than 90% of accuracy, it requires more than 900 seconds to finish the training. Considering that in a scenario where the trained classifier (or model) needs to be frequently retrained to suit the changing pattern of the data, then C5.0 might not be an ideal option due to its long training time. While in other occasions where the classification accuracy is critical, then the classifiers can be trained beforehand with a large amount of training data. For these occasions, Random Forest and C5.0 should be considered.

Undeniably, the testing/detection time plays an important role when the real-time detection capability is desired.

5. The deep learning algorithm was implemented using H2O which is a Java-based platform that is different from the implementation of other algorithms. Strictly speaking, the execution time is not comparable.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2710540, IEEE Access

TABLE 6: Algorithms Training/Detection Time on 200k Training/Testing Datasets Using 32 CPU Cores

Algorithms	Training (Seconds)	Detection (Seconds)
kNN	219.6	**741**
k-kNN	25.2	86.4
BLR	**3**	18.6
GBM	88.8	1.8
C5.0	**9504**	72.6
Random Forest	934.8	24
Naive Bayes	45.6	417
Neural Network	16.8	**1.2**
Deep Learning	27	22.2

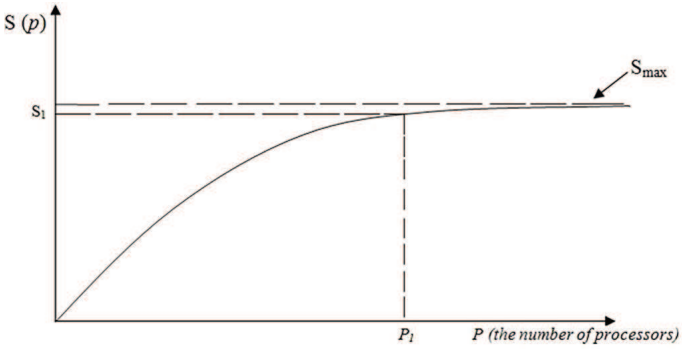


Fig. 7: The possible speedup trend – a linear speedup

Despite the selected algorithms spending considerably less time for detection compared with that in training, the detection process can still be time-consuming if the size of data is considerable.

As shown in Table. 6, kNN took the longest time to finish detecting, which was approximate 12 minutes. On the contrary, GBM and Neural Network only needed 1.8 and 1.2 seconds respectively. For C5.0, around 72 seconds was required, while for the same problem size, it took Random Forest less than half a minute. Therefore, for the applications that demand timely detection, then the GBM algorithm can be the preferred candidate due to its relatively high detection accuracy (81% of the TPR using 200k training) and instant detection time.

To boost the training and detecting process of all algorithms, a straightforward way is to add more computational resources, namely CPU cores in our scenario, to accelerate the computation. However, our scalability experiment has revealed that it is not always cost-effective to add more CPU cores to accomplish real-time detection capability, because the speedup of the majority of algorithms can achieve is non-liner. With the number of CPU cores increasing, the execution time will reduces continuously. But when the number of CPU cores reaches a certain number P_1, there will be no significant improvement gained no matter how many CPU cores are involved. (as shown in Fig. 7 which depicts a general speedup trend of all algorithms except fro deep learning).

One of the reasons that cause this phenomena is that with the number of CPU cores growing, the process running on each CPU core needs to spend time synchronizing the tasks. The other reason is related to the implement parallelism of R-based machine learning algorithms and the doParallel package. For example, some algorithms may not originally design to implement parallelism, only a certain stage of the calculation process (i.e. the for loop) can be run in parallel, while the majority of the calculation is processed in a sequential manner. Another reason is that the CPU cores is not fully utilized during the execution. All these possible factors hinder the effective use of parallel computational resources.

Noticeably, deep learning achieved a superlinear speedup on our shared-memory multi-processor workstation. Based on [21], superlinear speedup could happen due to reducing the number of clocks per instruction (CPI) for memory access in the parallel environment. There are several factors causing the reduction of the CPI for memory access. From the perspective of hardware, one of the factors is the shared cache provided by our shared-memory multi-processor system. The system is equipped with 2 physical Intel(R) Xeon(R) CPU E5-2690 v3 2.60GHz CPUs, each of which offers 12 CPU cores (two threads per core). Within one physical CPU, 24 logical CPU cores share the cache memory. According to [21], shared-cache processors provide more efficient memory access than processors use private cache. We believe that the deep learning speedup has benefited from the efficiency of shared-cache as can be seen from Fig.6-(a) that when using less than 16 CPU cores, there was a sharp increase of the speedup trend, because all the 16 processes can be run on one physical CPU sharing cache memory. When doubling the number CPU cores from 16 to 32, there was still an increase of the speedup, but the increase rate was not as sharp as previous, because 32 processes have to run on 2 physical CPUs which are not sharing cache. Two groups of processes running on 2 physical CPUs have to share the main memory which is less efficient than sharing the cache. Besides, doubling the number of processes also increases the amount of communications which might cause extra performance overheads.

In algorithm level, deep learning implemented by H2O is optimized for lock-free parallelization scheme, which minimizes the overhead caused by memory locking [4] [20]. With a parallelized version of stochastic gradient descent (SGD) utilizing Hogwild! [20], the H20 platform enables processors to have free access to shared memory, allowing processors to update their own memory components without causing overwrites. Hence, no locking is needed for each processor, which greatly increases the efficiency of inter-process communication, thus contributing to a better speedup.

Normally, a good scalability reflects that implementing parallelism can significantly shorten the execution time and indicates that there is much room for performance improvement. Based on our experiment, by referring to the speedup trends of the algorithms shown in Fig.6, we can draw a conclusion that apart from Deep Learning, using more than 16 CPU cores is not able to significantly accelerate the training process with no more than 200k training datasets. But the speedup trend of Deep Learning shows that it is still hungry for computation. Using more than 32 CPU cores can still lead to a moderate decrease in its training time, which makes real-time training achievable by using Deep Learning.

8 Conclusion and future work

In conclusion, we collected 9 mainstream machine learning algorithms and studied them in terms of classification performance, stability and scalability based on tweets datasets. We applied these algorithms under different scenarios by varying the volumes of training data and the ratio of spam to non-spam to evaluate their performance of detecting Twitter spams in terms of accuracy, the TPR/FPR and the F-measure. We also investigated the performance stability of each algorithm to understand how random sampling and variation of testing data affect the detection performance. The outcome of our experiment shows that Random Forest and C5.0 stood out due to their superior detection accuracy, and Random Forest performed more stably than other algorithms. To understand the cost-effectiveness of selected algorithms in a parallel environment when processing a large amount of datasets, we examine the scalability of each algorithm using different number of CPU cores, aiming to observe how parallel computational resources impact the algorithms' training process and how much resource is suitable for a certain problem size. Based on our experiment, we observed a superlinear speedup trend achieved by Deep Learning, showing that it is capable of effectively utilizing parallel resources and possible to accomplish real-time training and testing tasks.

There are problems worthy of further study in future. It will be interesting to evaluate whether the performance of these algorithms can be further improved by using more tweets for training. For the scalability tests, we would like to carry out experiments on laboratory-size computer clusters to explore whether the algorithms such as Deep Learning and Random Forest would achieve better scalability and performance in a large scale of parallel environment.

Acknowledgement

Guanjun Lin is supported by the Australian Government Research Training Program Scholarship. And this work was supported by the National Natural Science Foundation of China (No. 61401371).

References

[1] F. Benevenuto, G. Magno, T. Rodrigues, and V. Almeida, "Detecting spammers on twitter," in *Collaboration, electronic messaging, anti-abuse and spam conference (CEAS)*, vol. 6, 2010, p. 12.

[2] G. Biau, "Analysis of a random forests model," *Journal of Machine Learning Research*, vol. 13, no. Apr, pp. 1063–1095, 2012.

[3] C. M. Bishop, "Pattern recognition and machine learning," *Machine Learning*, vol. 128, 2006.

[4] A. Candel, V. Parmar, E. LeDell, and A. Arora, "Deep learning with h2o," http://h2o2016.wpengine.com/wp-content/themes/h2o2016/images/resources/DeepLearningBooklet.pdf, Oct 2016.

[5] C. Chen, J. Zhang, X. Chen, Y. Xiang, and 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.

[6] D. Conway and J. White, *Machine learning for hackers*. " O'Reilly Media, Inc.", 2012.

[7] M. Egele, G. Stringhini, C. Kruegel, and G. Vigna, "Compa: Detecting compromised accounts on social networks." in *NDSS*, 2013.

[8] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: a statistical view of boosting," *Annals of Statistics*, vol. 28, p. 2000, 1998.

[9] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[10] A. K. Ghosh, P. Chaudhuri, and C. Murthy, "On visualization and aggregation of nearest neighbor classifiers," *IEEE transactions on pattern analysis and machine intelligence*, vol. 27, no. 10, pp. 1592–1602, 2005.

[11] H2O.ai, "H2o for business," http://www.h2o.ai//, 2016, accessed: 2016-08-29.

[12] K. Hechenbichler and K. Schliep, "Weighted k-nearest-neighbor techniques and ordinal classification," 2004.

[13] M. Kuhn, "Caret package," *Journal of Statistical Software*, vol. 28, no. 5, 2008.

[14] ——, "A short introduction to the caret package," *R Project website. cran. r-project. org/web/packages/caret/vignettes/caret. pdf. Published August*, vol. 6, 2015.

[15] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[16] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic, "Deep learning applications and challenges in big data analytics," *Journal of Big Data*, vol. 2, no. 1, p. 1, 2015.

[17] J. Oliver, P. Pajares, C. Ke, C. Chen, and Y. Xiang, "An in-depth analysis of abuse on twitter," *Trend Micro*, vol. 225, 2014.

[18] C. Pash, "The lure of naked hollywood star photos sent the internet into meltdown in new zealand," http://www.businessinsider.com.au/the-lure-of-naked-hollywood-star-photos-sent-the-internet-into-meltdown-in-new-zealand-2014-9, September 2014, accessed: 2016-08-06.

[19] R. Quinlan, "Data mining tools see5 and c5. 0," 2004.

[20] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in Neural Information Processing Systems*, 2011, pp. 693–701.

[21] S. Ristov, R. Prodan, M. Gusev, and K. Skala, "Superlinear speedup in hpc systems: Why and when?" in *Computer Science and Information Systems (FedCSIS), 2016 Federated Conference on*. IEEE, 2016, pp. 889–898.

[22] J. Shan, "Superlinear speedup in parallel computation," *CCS, Northeastern Univ., Massachusetts, Course Report*, 2002.

[23] J. Song, S. Lee, and J. Kim, "Spam filtering in twitter using sender-receiver relationship," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2011, pp. 301–317.

[24] Statista, "Number of monthly active twitter users worldwide from 1st quarter 2010 to 2nd quarter 2016 (in millions)," http://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/, 2016, accessed: 2016-08-09.

[25] G. Stringhini, C. Kruegel, and G. Vigna, "Detecting spammers on social networks," in *Proceedings of the 26th Annual Computer Security Applications Conference*. ACM, 2010, pp. 1–9.

[26] K. Thomas, C. Grier, J. Ma, V. Paxson, and D. Song, "Design and evaluation of a real-time url spam filtering service," in *2011 IEEE Symposium on Security and Privacy*. IEEE, 2011, pp. 447–462.

[27] C. Vecchiola, S. Pandey, and R. Buyya, "High-performance cloud computing: A view of scientific applications," in *2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks*. IEEE, 2009, pp. 4–16.

[28] A. H. Wang, "Don't follow me: Spam detection in twitter," in *Security and Cryptography (SECRYPT), Proceedings of the 2010 International Conference on*. IEEE, 2010, pp. 1–10.

[29] D. Wang, S. B. Navathe, L. Liu, D. Irani, A. Tamersoy, and C. Pu, "Click traffic analysis of short url spam on twitter," in *Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom), 2013 9th International Conference Conference on*. IEEE, 2013, pp. 250–259.

[30] S. Weston and R. Calaway, "Getting started with doparallel and foreach," *Available on https://cran. r-project. org/web/packages/doparallel/vignettes/gettingstartedparallel. pdf*, 2015.

[31] C. Yang, R. Harkreader, and G. Gu, "Empirical evaluation and new design for fighting evolving twitter spammers," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 8, pp. 1280–1293, 2013.

[32] C. Yang, R. C. Harkreader, and G. Gu, "Die free or live hard? empirical evaluation and new design for fighting evolving twitter spammers," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2011, pp. 318–337.

[33] E. J. H. Yero and M. A. A. Henriques, "Speedup and scalability analysis of master–slave applications on large heterogeneous clusters," *Journal of Parallel and Distributed Computing*, vol. 67, no. 11, pp. 1155–1167, 2007.

[34] K. Zetter, "Trick or tweet? malware abundant in twitter urls," 2009.

[35] X. Zhang, S. Zhu, and W. Liang, "Detecting spam and promoting campaigns in the twitter social network," in *2012 IEEE 12th International Conference on Data Mining*. IEEE, 2012, pp. 1194–1199.

[36] J. Zorbas, D. Reble, and R. VanKooten, "Measuring the scalability of parallel computer systems," in *Proceedings of the 1989 ACM/IEEE conference on Supercomputing*. ACM, 1989, pp. 832–841.