Machine learning techniques for web intrusion detection – a comparison

Truong Son Pham
Le Quy Don technical University
Faculty of information technology
Hanoi, Vietnam
sonpham.mta@gmail.com

Tuan Hao Hoang
Le Quy Don technical University
Faculty of information technology
Hanoi, Vietnam
haoth@gmail.com

Van Canh Vu
Le Quy Don technical University
Faculty of information technology
Hanoi, Vietnam
canhvuvan@yahoo.com

***Abstract*—The rapid development of web applications has created many security problems related to intrusions not just on computer, network systems, but also on web applications themselves. In Web Intrusion Systems (WIS), most techniques used nowadays are not able to deal with the dynamic and complex nature of cyber-attacks on web applications and related issues. However, web intrusion techniques based on machine learning approaches with statistical analysis of data enable autonomous detect intrusive and non-intrusive traffic with low false-positive errors. In this paper, we present the survey of various machine learning techniques used to build WIS. In addition, we develop the experimental framework for comparative analysis of some machine learning techniques applying on the well-known benchmark data set – CSIC 2010 HTTP [13].**

Keywords*— *web attacks; intrusion detection system; anomaly intrusion detection; web application security; machine learning techniques comparison.

I. Introduction

The internet, nowsaday, is an important part of modern life. Thanks to web applications, we are just a click away from the rest of the world. However, like other software applications, web applications have been developed and deployed with mininal attention given to security risks, resulting in surprising number of corporate sites that are vulnerable to hackers. For instance, modern web applications have a tendency to utilize third party API:s, services and this can be seen as additional room for security vulnerabilities. Over the past few years the amount of intrusion attempts has been on a steady increase and statistics show that as much as 75% of the cyber attacks [5] target web applications specifically. In a survey conducted [5] it was found that web attacks cost organizations over 100 times more than malware, and 50 times more than viruses, worms and trojans annually.

Intrusion detection plays an important role in addressing the security problems of web servers, by providing timely identification of malicious activity and supporting effective response to attacks. Unfortunately, detection of attacks has been performed by applying simple pattern-matching techniques to the contents of HTTP requests or by identifying trends in a large set of web-related events. In addition, most intrusion detection systems focus on a single event stream, such as the network traffic directed to a server host or the access logs produced by a server application. The lack of a state-full detection model and the inability to analyze different event streams in an integrated way severely limits the effectiveness of current intrusion detection approaches. To improve the detection of web-based attacks, we propose an integrated approach that performs intrusion detection using state-full analysis of multiple event streams. The approach is centered-around the State-Transition Analysis Technique (STAT) [4], which supports the modeling of multi-step, attacks complex in terms of states and transitions. STAT-based intrusion detection systems can be developed in a modular fashion, by extending an application in dependent runtime with components that deal with specific application domains.

One way to potentially keep up with the antivirus avoidance techniques used in modern intrusion is to augment existing detection systems with machine learning classifiers. Machine learning classification algorithms facilitate construction of classifiers, which automatically learn the characteristics of each class, such as intrusion and benign files, by learning from example data. This kind of approach, while still deploying intrusion detection system, offers the promise of increased robustness in the face of newly adapted threats that are slightly modified, but retain some of the characteristics of past intrusion.

To use machine learning to classify executable files as intrusion or benign, one must first build labeled datasets for training. A set of records-one for each benign and intrusion file comprises the database. Each record contains a set of features and one label (also referred to as the class). The features of each file are derived from some specific characteristics of the file, such as the size of the file or the frequency of a certain code snippet in the file; the label is a binary value indicating whether or not the file is intrusion. Learning algorithms analyze records designated for training to generate a mathematical model that maps the relationship of file features and labels. That model, which is called the classifier, is used to predict the class of each record in the test data, or the records designated for testing. The classifier cannot read the labels when making predictions; test data labels are only used when the predictions are compared against the true labels in subsequent analysis of performance.

A variety of features have been studied as potentially effective discriminators between intrusion and benign software. These features can be extracted either through dynamic or static analysis. Static analysis extracts features either from the header, which contains metadata, or the body, which contains the actual code and data, of an executable file. Dynamic analysis involves carefully executing a program in a safe environment and measuring features by recording behavior such as interactions with the operating system or network traffic. While dynamic analysis can sometimes reveal behavior in intrusion that would be difficult to extract from static analysis [2], dynamic analysis is also more resource intensive than some types of static analysis [3], and can currently be defeated in a variety of ways [1].

Recently, numbers of anomaly computer and network intrusion detection systems are developed based on many different machine learning techniques with the hope of improving detection rates and adaptability. For example, Laskov et al. [15] experiment some machine learning techniques included K-Nearest Neighbor, Multi-layer Perceptron, Support Vector machine to improve the generalization of intrusion detection system to new malicious patterns. Sommer and Paxson [16] propose the interesting idea that machine learning techniques supported anomaly detection is better for finding variations of known patterns rather than previously unknown malicious activity.

The paper is conducted as follow. After the Introduction chapter, chapter II describes some background in web vulnerabilities. Chapter III introduces some machine learning techniques that are normally used for web intrusion detection. The paper then describes some overview of web vulnerabilities system in chapter IV. Chapter V presents some experiments and results conducted in this paper. Conclusions and future works are shown in the last chapter.

II. WEB VULNERABILITIES

There is a large amount of web vulnerabilities. In this subsection the most important ones are explained, considering the frequency of exploitation and the impact. The attacks listed are included in some way in the Owasp Top Ten Project [6][7], which presents the most critical web application security vulnerabilities. Next, a description of the most relevant web vulnerabilities is presented.

Injection: Injection flaws, such as SQL, OS, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

Broken Authentication and Session Management: Account credentials and session tokens are often not properly protected. Attackers compromise passwords, keys, or authentication tokens to assume other users' identities.

Cross site scripting (XSS): XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

Insecure Direct Object Reference: A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.

Web application and server misconfiguration: Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. Secure settings should be defined, implemented, and maintained, as defaults are often insecure. Additionally, software should be kept up to date.

Sensitive Data Exposure: Many web applications do not properly protect sensitive data, such as credit cards, tax IDs, and authentication credentials. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.

Broken Access control: Most web applications verify function level access rights before making that functionality visible in the UI. However, applications need to perform the same access control checks on the server when each function is accessed. If requests are not verified, attackers will be able to forge requests in order to access functionality without proper authorization.

Cross Site Request Forgery (CSRF): A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim.

Using Components with Known Vulnerabilities: Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts.

Invalidated Redirects and Forwards: Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

Attacks exploiting these vulnerabilities will be used to test the performance of the Web applications frequently being presented, thus these attacks are included in the malicious traffic generated to test the system. Some machine learning techniques is explained in Sec. III and overview of Web intrusion detection will be described in Sec. IV. Some experiments and results for Web-based intrusion detection are listed in Sec. V. and conclusions and future work will be discussed in the last Sec. VI.

III. Machine learning techniques

Machine learning is a subfield of computer science that evolved from the study of pattern recognition and computational learning theory in artificial intelligence. In 1959, Arthur Samuel defined machine learning as a "Field of study that gives computers the ability to learn without being explicitly programmed". Machine learning explores the study and construction of algorithms that can learn from and make predictions on data. Such algorithms operate by building a model from an example training set of input observations in order to make data-driven predictions or decisions expressed as outputs, rather than following strictly static program instructions.

The following sections will list and descript different machine learning techniques, which are used for classification problems, and will be compared in this paper.

A. Random forest (RdF)

Breiman (2001) proposed random forests, which add an additional layer of randomness to bagging. In addition to constructing each tree using a different bootstrap sample of the data, random forests change how the classification or regression trees are constructed. In standard trees, each node is split using the best split among all variables. In a random forest, each node is split using the best among a subset of predictors randomly chosen at that node. This somewhat counterintuitive strategy turns out to perform very well compared to many other classifiers, including discriminant analysis, support vector machines and neural networks, and is robust against overfitting [15]. In addition, it is very user-friendly in the sense that it has only two parameters (the number of variables in the random subset at each node and the number of trees in the forest), and is usually not very sensitive to their values.

B. Logistic regression (LRg)

Logistic regression has a long tradition with widely varying applications such as modeling dose-response data and purchase-choice data. Unfortunately, many introductory statistics courses do not cover this fairly simple method. Many texts in categorical statistics cover it [18], in addition to texts on logistic regression [17]. Some analysts use the method with a different distribution function, the normal. In that case, it is called probit analysis. Some analysts use discriminant analysis instead of logistic regression because they prefer to think of the continuous variables as Ys and the categories as Xs and work backwards. However, discriminant analysis assumes that the continuous data are normally distributed random responses, rather than fixed repressors.

Nominal logistic regression estimates the probability of choosing one of the response levels as a smooth function of the x factor. The fitted probabilities must be between 0 and 1, and must sum to 1 across the response levels for a given factor value.

In a logistic probability plot, the y-axis represents probability. For k response levels, k - 1 smooth curves partition the total probability (which equals 1) among the response levels. The fitting principle for a logistic regression minimizes the sum of the negative natural logarithms of the probabilities fitted to the response events that occur (that is, maximum likelihood).

When Y is ordinal, a modified version of logistic regression is used for fitting. The cumulative probability of being at or below each response level is modeled by a curve. The curves are the same for each level except that they are shifted to the right or left.

The ordinal logistic model fits a different intercept, but the same slope, for each of $r-1$ cumulative logistic comparisons, where r is the number of response levels. Each parameter estimate can be examined and tested individually, although this is seldom of much interest.

The ordinal model is preferred to the nominal model when it is appropriate because it has fewer parameters to estimate. In fact, it is practical to fit ordinal responses with hundreds of response levels.

C. Decision tree (DTr)

A decision tree is a classifier expressed as a recursive partition of the instance space. The decision tree consists of nodes that form a rooted tree, meaning it is a directed tree with a node called “root” that has no incoming edges. All other nodes have exactly one incoming edge. A node with outgoing edges is called an internal or test node. All other nodes are called leaves (also known as terminal or decision nodes). In a decision tree, each internal node splits the instance space into two or more sub-spaces according to a certain discrete function of the input attributes values. In the simplest and most frequent case, each test considers a single attribute, such that the instance space is partitioned according to the attribute’s value. In the case of numeric attributes, the condition refers to a range.

Each leaf is assigned to one class representing the most appropriate target value. Alternatively, the leaf may hold a probability vector indicating the probability of the target attribute having a certain value. Instances are classified by navigating them from the root of the tree down to a leaf, according to the outcome of the tests along the path. Internal nodes are represented as circles, whereas leaves are denoted as triangles. Note that this decision tree incorporates both nominal and numeric attributes. Given this classifier, the analyst can predict the response of a potential customer (by sorting it down the tree), and understand the behavioral characteristics of the entire potential customers population regarding direct mailing. Each node is labeled with the attribute it tests, and its branches are labeled with its corresponding values [16].

D. AdaBoostClassifier (ABc)

Working in Valiant's PAC (probably approximately correct) learning model [25], Kearns and Valiant [22] were the first to pose the question of whether a "weak" learning algorithm that performs just slightly better than random guessing can be "boosted" into an arbitrarily accurate "strong" learning algorithm. Schapire [23] came up with the first provable polynomial-time boosting algorithm in 1989. A year later, Freund [19] developed a much more efficient boosting algorithm which, although optimal in a certain sense, nevertheless suffered like Schapire's algorithm from certain practical drawbacks. The first experiments with these early boosting algorithms were carried out by Drucker, Schapire and Simard [20] on an OCR task. The AdaBoost algorithm, introduced in 1995 by Freund and Schapire [21], solved many of the practical difficulties of the earlier boosting algorithms, and is the focus of this paper. Pseudocode for AdaBoost is given in Fig. 1 in the slightly generalized form given by Schapire and Singer [24]. The algorithm takes as input a training set $(x_1, y_1)...(x_m, y_m)$ where each x_i belongs to some domain or instance space X, and each label y_i is in some label set Y.

Assume that $Y = \{-1, +1\}$; AdaBoost calls a given weak or base learning algorithm repeatedly in a series of rounds $t = 1, ..., T$. One of the main ideas of the algorithm is to maintain a distribution or set of weights over the training set. The weight of this distribution on training example i on round t is denoted $D_t(i)$. Initially, all weights are set equally, but on each round, the weights of incorrectly classified examples are increased so that the base learner is forced to focus on the hard examples in the training set. The base learner's job is to find a base classifier $h_t : X \Rightarrow \mathbb{R}$ appropriate for the distribution D_t. In the simplest case, the range of each h_t is binary, i.e., restricted to $\{-1, +1\}$; the base learner's job then is to minimize the error

E. SGDClassifier (SGDc)

The stochastic gradient descent (SGD) algorithm is a drastic simplification. Instead of computing the gradient of $E_n(f_w)$ exactly, each iteration estimates this gradient on the basis of a single randomly picked example z_t:

$$\mathrm{w}_{t+1} = w_t - \gamma_t \nabla_w Q(z_t, w_t)$$

The stochastic process $\{w_t, t = 1, ...\}$ depends on the examples randomly picked at each iteration. It is hoped that behaves like its expectation despite the noise introduced by this simplified procedure. Since the stochastic algorithm does not need to remember which examples were visited during the previous iterations, it can process examples on the fly in a deployed system. In such a situation, the stochastic gradient descent directly optimizes the expected risk, since the examples are randomly drawn from the ground truth distribution. The convergence of stochastic gradient descent has been studied extensively in the stochastic approximation literature. Convergence results usually require decreasing gains satisfying the conditions $\sum_t \gamma_t^2 \prec \infty$ and $\sum_t \gamma_t = \infty$.

The Robbins-Siegmund theorem provides the means to establish almost sure convergence under mild conditions [26], including cases where the loss function is not everywhere differentiable. The convergence speed of stochastic gradient descent is in fact limited by the noisy approximation of the true gradient. When the gains decrease too slowly, the variance of the parameter estimate w_t decreases equally slowly. When the gains decrease too quickly, the expectation of the parameter estimate wt takes a very long time to approach the optimum. Under sufficient regularity condition, the best convergence speed is achieved using gains $\gamma_t \sim t^{-1}$. The expectation of the residual error then decreases with similar speed that is $\mathbb{E}_p \sim t^{-1}$.

The second order stochastic gradient descent (2SGD) multiplies the gradients by a positive definite matrix Γt approaching the inverse of the Hessian:

$$\mathrm{w}_{t+1} = w_t - \gamma_t \Gamma_t \nabla_w Q(z_t, w_t)$$

Unfortunately, this modification does not reduce the stochastic noise and therefore does not significantly improve the variance of wt. Although constants are improved, the expectation of the residual error still decreases like t^{-1}, that is, $\mathbb{E}_p \sim t^{-1}$.

IV. WEB INTRUSION DETECTION SYSTEM OVERVIEW

This section presents an overview of our machine learning based IDS for web intrusion detection. Network-based intrusion detection system (NIDS) has played an important role in network security due to the widespread use of computer network, the increase in valuable resources and the rapid development of attackers [8]. Traditionally, IDS's have been classified as either signature detection systems (also called negative approach) or anomaly detection systems (positive approach). Nevertheless, traditional signature-based intrusion detection techniques have failed to fully protect networks and systems from increasingly sophisticated attacks and malwares. Consequently, machine learning based intrusion detection systems have become an indispensable component of security infrastructure used to detect these threats before they inflict widespread damages [9]. In order to compare different machine learning techniques for intrusion detection, we implemented a machine learning based IDS and captured LAN traffic for analysis using monitoring feature of Cisco device: Switched Port Analyzer (SPAN). A diagram of our test environment is shown in Fig. 1.

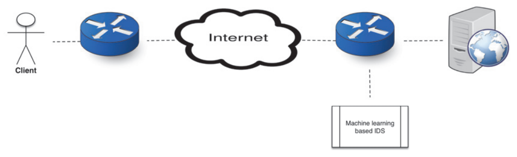

Figure 1: Capture LAN traffic for IDS on monitoring port of network switch

A. Achitecture

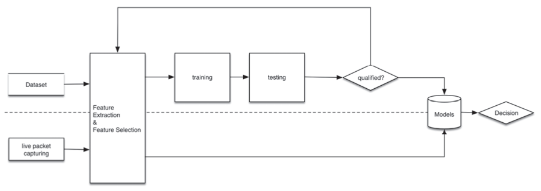

Figure 2: Web Intrusion Detection System Architecture

Fig. 2 shows the architecture of our web intrusion detection system, which has been divided into two parts, one for training and validation of the machine learning techniques and the other one for live packet capturing on network card on a SPAN port of a switch, which is directly connected to the web server. These packets will be "in real time" classified by the machine learning models, which have been qualified in the training process.

We also used some feature extraction and feature selection techniques in order to gain more information from the text features and also improve the performance of the IDS.

B. Traning and validation process

In this phase, a dataset has been used to train and test the machine learning models. The dataset itself contains more than one text feature, which are not able to be fed to the machine learning algorithms, because almost the algorithms expect numerical features [28].
In order to solve this problem, some techniques are provided, which allow us to extract the numerical features from text content [12], namely:

- "Tokenizing strings and giving an integer id for each possible token, for instance by using white-spaces and punctuation as token separators.
- Counting the occurrences of tokens in each document.
- Normalizing and weighting with diminishing importance tokens that occur in the majority of samples / documents" [28].

In this dataset, features and samples are defined as follows:

- Each column of the dataset is considered a feature.
- Each row, also called as a vector is treated as a sample.

In our systems, we used the combination of these three feature extraction approaches, which means tokenization and occurrence counting followed by re-weighting the count features into floating point values suitable for usage by a classifier [28].

The machine learning models will be trained until they privilege by fulfilling a necessary condition in the validation phase.

C. Live packet capturing

In order to detect an attack in real time, it wouldn't make sense to look into the log file of a webserver after the attacks. Therefore, we have implemented a live packet capturing module for our system. Which works directly on the network card, captures the packets and sends it to the detection process. We use the same capturing technique, which has been applied in Wireshark, but rewritten in python with only the necessary functions.

D. Detection process

In this phase, a network packet, which has been captured by the live capturing module, will be classified with the models, which are trained and qualified in the training/validation process.

After the machine learning models fulfilled all necessary conditions, we have to rely on the decision of these models. We also used a weighted voting classifier to improve the accuracy of our IDS.

In the next section, we will test the machine learning methods, which are descripted in section 3 and discuss the test results.

V. EXPERIMENTS AND RESULTS

This section presents the settings for the experiments in this paper and the experimental results of applying different machine learning techniques to the CSIC 2010 HTTP dataset [13]. To apply learning methods to these data sets, we used the implementations in Scikit-learn.

A. Experimental settings

In order to compare different machine learning techniques, an experiment was set up. This aims to examine the impact of the dataset on these techniques, to see how good these techniques could work with these data. The dataset contains 223585 samples of which there are 119585 web attacks labeled as "anom" and 104000 labeled as "norm". We divided this dataset into two parts, one for training and one for validation.

TABLE 1. BEST PERFORMANCE OF LOGISTIC REGRESSION

Label	Training Set	Validation Set
norm	69705	34295
anom	80096	39489

B. Results

The performance of each algorithm is measured by the precision, recall and f1-score on the testing data set. They are shown in the following tables and figures.

TABLE 2. BEST PERFORMANCE OF LOGISTIC REGRESSION

Methods		RdF	LRg	DTr	ABc	SGDc
anom	**precision**	79.70	99.39	88.10	67.24	72.45
	recall	87.11	93.05	88.28	89.19	92.04
	F1 score	83.24	96.11	88.19	76.68	81.08
norm	**precision**	83.37	92.54	86.48	80.06	86.69
	recall	74.46	99.34	86.26	49.98	59.70
	F1 score	78.67	95.82	86.37	61.54	70.71

Table 2 shows the test results of the learning methods. They are given in three terms, namely: precision, recall and f1-score. In this case, precision is "how useful the search results are", recall is "how complete the results are", and the f1-score is a measure of the test's accuracy, it is the harmonic mean of precision and recall.

It can be seen from this table that Logistic Regression represents both highest precision and highest recall, where high precision relates to a low false positive rate, and high recall relates to a low false negative rate. Logistic Regression is good in this case for some reasons. Firstly, here is a binary classification while Logistic Regression is a binary classifier. Secondly the raw data have been transformed with a feature extraction method, which combines all the options of Tokenizing, Counting and Normalizing in a single model, the result of this feature extraction model is a sparse matrix, which is a very “well-liked” input for Logistic Regression. Lastly, not like other Machine learning methods, Logistic regression assumes that there is only one decision boundary that is smooth and non-linear and Logistic regression is intrinsically simple, it has low variance and so is less prone to over-fitting [27].

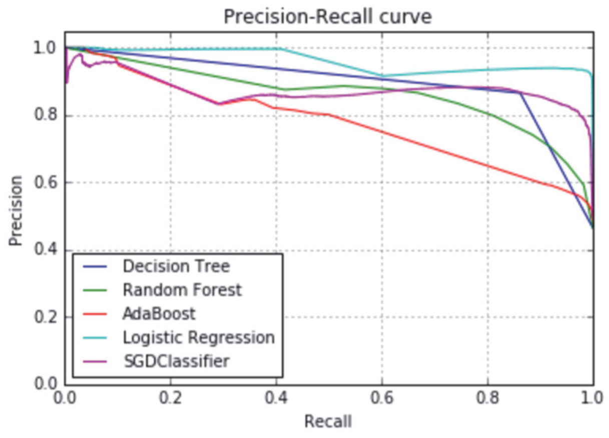

Figure 3. Precision recall curves of learning methods

Figure 3 shows the precision recall curve of the machine learning methods on this dataset. “A system with high recall but low precision returns many results, but most of its predicted labels are incorrect when compared to the training labels. A system with high precision but low recall is just the opposite, returning very few results, but most of its predicted labels are correct when compared to the training labels. An ideal system with high precision and high recall will return many results, with all results labeled correctly” [30]. In this case Logistic Regression represents a very good learning method on this dataset.

After we saw that Logistic Regression is a best learning method for the CSIC 2010 HTTP dataset. It would make sense that we look deeper into this model to improve its performance. To do that, we used a scoring function to measure the accuracy of the classifiers with very different values of parameters. To find out whether a classifier is overfitting or under-fitting, it is consistently advantageous to see the effect of a parameter on the training and the validation score [31]. In this case we plotted the validation curve of the Logistic Regression classifier with different C values, which represents the inverse of regularization strength. “If the training score and the validation score are both low, the estimator will be under-fitting. If the training score is high and the validation score is low, the estimator is overfitting and otherwise it is working very well” [31].

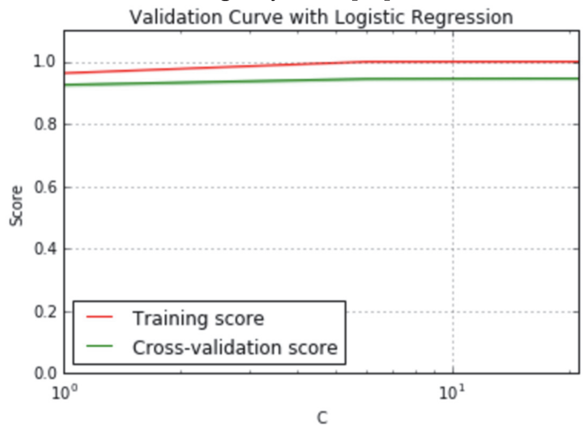

Figure 4. Validation curve of Logistic Regression

Figure 4 shows that Logistic Regression is neither overfitting nor under-fitting.

We can improve the accuracy of a classifier not only by turning its parameters but also by choosing, which is the best amount of training samples. A huge amount of training sample doesn’t mean automatically a good validation score. In order to find out how much advantages we get from adding more training data a learning curve will be plotted. Both the validation and the training score should converge to a value that is high with growing amount of the training samples so that we will benefit from more training data [31].

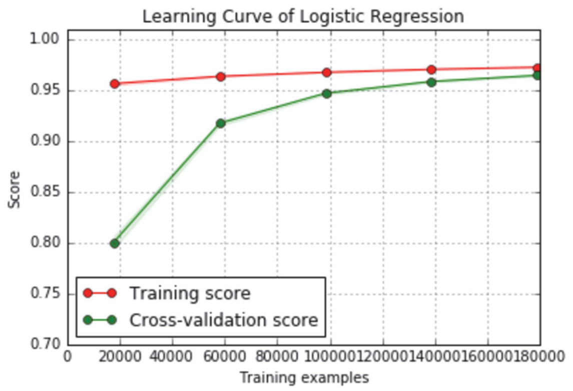

Figure 5. Learning curve of Logistic Regression

From the figure 5 it can be seen that the Logistic Regression could be improved from more training examples. After all these experiments we are able to improve the performance of Logistic Regression by using feature selection techniques and also parameter turning. For feature selection we removed all the feature with low variance, selected the best number of features with recursive feature elimination, which was 77558. For parameter turning, after a lot of experiments we found out that the best inverse of regularization strength for Logistic Regression in this case is 13.0.

TABLE 3. BEST PERFORMANCE OF LOGISTIC REGRESSION

	Norm	Anom
Precision	94.36	99.75
Recall	99.73	94.82
F1 Score	96.97	97.22

Table 3 represents the best test results of Logistic Regression in our experiments.

VI. CONCLUSION AND FUTURE WORK

In this paper, we compared different machine learning techniques on web intrusion detection. In the experiments we used CISC 2010 HTTP dataset, which includes attacks such as SQL injection, buffer overflow, information gathering, files disclosure, CRLF injection, XSS, server side include, parameter tampering and so on. Experiments showed that Logistic Regression is the best learning methods for this problem. Logistic Regression represents a very good performance with the highest recall and also highest precision. We have also tried to improving its performance with different feature extraction, feature selection and also parameter turning techniques. The results looked better afterwards.

There are several potential researches that are arisen from this paper. First and very important one is to evaluate the effectiveness of the proposed methods on more data sets particularly the data set was collected from the real environment as in [14]. Second, we could try to test and compare these techniques on unseen attacks to see if these attacks could be detected, maybe it would make sense to generate some artificial attack data to improve the performance of IDS as in [10]. In the future, we are planning to do this.

References

[1] E. Nasi, "Bypass Antivirus Dynamic Analysis: Limitations of the AV model and how to exploit them," Aug. 2014.

[2] K. Rieck, T. Holz, and C. Willems, "Learning and classification of malware behavior," Proc. 5th Int. Conf. Detect. Intrusions Malware, Vulnerability Assess. (DIMVA '08), 2008.

[3] G. Yan, N. Brown, and D. Kong, "Exploring Discriminatory Features for Automated Malware Classification," pp. 41–61, 2013.

[4] K. Ilgun, R.A. Kemmerer, and P.A. Porras. State Transition Analysis: A Rule-Based Intrusion Detection System. IEEE Transactions on Software Engineering, 21(3):181–199, March 1995.

[5] Imperva, Web Security Threats, http://www.imperva.com/docs/DS_Web_Security_Threats.pdf, 2013-03-10.

[6] Top 10 2013. Owasp Foundation. https://www.owasp.org/index.php/Top_10_2013-Top_10

[7] Banković Z., Moya J.M., Araujo A., Bojanić S., NietoTaladriz O.: A Genetic Algorithm-based Solution for Intrusion Detection. Journal of Information Assurance and Security, 4, 192-199 (2009)

[8] W. Lee, S. Stolfo, and K. Mok. A data mining framework for building intrusion detection models. In Proceedings of the 1999 IEEE Symposium on Security and Privacy (SSP '99), pages 120–132, Washington - Brussels - Tokyo, 1999. IEEE.

[9] C.-F. Tsai, Y.-F. Hsu, C.-Y. Lin, and W.-Y. Lin. Intrusion detection by machine learning: A review. Expert Systems with Applications, 36(10):11994–12000, 2009.

[10] T. S. Pham, Q. U. Nguyen and X. H. Nguyen. Generating Artificial Attack Data for Intrusion Detection Using Machine Learning. SoICT '14 Proceedings of the Fifth Symposium on Information and Communication Technolog, pages 286-291, Hanoi, 2014. IEEE.

[11] Ed. Charu Aggarwa, Data Classification: Algorithms and Applications

[12] David D. Lewis, Feature Selection and Feature Extract ion for Text Categorization, HLT '91 Proceedings of the workshop on Speech and Natural Language Pages 212-217, Chicago, 1992

[13] CSIC 2010 HTTP dataset avaiable at: http://www.isi.csic.es/dataset/

[14] V. L. Cao, V. T. Hoang, and Q. U. Nguyen. A scheme for building a dataset for intrusion detection systems. In the 2013 Third World Congress on Information and Communication Technologies, pages 120–132, Hanoi- Vietnam, 2013. IEEE

[15] L. Breiman. Random forests. Machine Learning, 45(1): 5–32, 2001.

[16] Quinlan, J.R., Simplifying decision trees, International Journal of ManMachine Studies, 27, 221-234, 1987

[17] Hosmer, D.W., & Lemeshow, S. Applied Logistic Regression 1989

[18] Agresti, A. and Coull, B. A., "Approximate is better than "exact" for interval estimation of binomial proportions", *The American Statistician*, 1998, 119-126.

[19] Yoav Freund. Boosting a weak learning algorithm by majority. Information and Computation, 121(2):256–285, 1995

[20] Harris Drucker, Robert Schapire, and Patrice Simard. Boosting performance in neural networks. International Journal of Pattern Recognition and Artificial Intelligence, 7(4):705–719, 1993

[21] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences, 55(1):119–139, August 1997

[22] Michael Kearns and Leslie G. Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. Journal of the Association for Computing Machinery, 41(1):67–95, January 1994

[23] Robert E. Schapire. The strength of weak learnability. Machine Learning, 5(2):197– 227, 1990

[24] Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. Machine Learning, 37(3):297–336, December 1999.

[25] L. G. Valiant. A theory of the learnable. Communications of the ACM, 27(11):1134– 1142, November 1984.

[26] BOTTOU, L. and BOUSQUET, O. (2008): The Tradeoffs of Large Scale Learning, In Advances in Neural Information Processing Systems, vol.20, 161-168.

[27] Jack Rae, Google DeepMind Research Engineer , https://www.quora.com/What-are-the-advantages-of-logistic-regression-over-decision-trees.

[28] Text feature extraction scikit learn , http://scikit-learn.org/stable/modules/feature_extraction.html

[29] Machine learning in python scikit learn , http://scikit-learn.org/stable/index.html

[30] Precision recall python scikit learn , http://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html

[31] Validation curves scikit learn, http://scikit-learn.org/stable/modules/learning_curve.html