Deep Learning for Network Intrusion Detection: An Empirical Assessment

Arnaldo Gouveia[1,2] and Miguel Correia[2]

[1] IBM Belgium – Belgium
[2] INESC-ID, Instituto Superior Técnico, Universidade de Lisboa – Portugal

Abstract. The detection of security-related events using machine learning approaches has been extensively investigated in the past. Particularly, machine learning-based network intrusion detection has attracted a lot of attention due to its potential to detect unknown attacks. A number of classification techniques have been used for that purpose, but they were mostly classical schemes like decision trees. In this paper we go one step further and explore the use of a set of machine learning techniques denominated generically as "deep learning" that have been generating excellent results in other areas. We compare three recent techniques – generalized linear models, gradient boosting machines, and deep learning – with classical classifiers. The comparison is performed using a recent data set of network communication traces designed carefully for evaluating intrusion detection schemes. We show that deep learning techniques have an undeniable value over older algorithms, since better model fitting indicators can be achieved.

Keywords: Network Security, Network Intrusion Detection Systems, Machine Learning, Deep Learning, NIDS Performance Evaluation, Data Mining

1 Introduction

With the continuous growth in number and impact of network attacks, *network intrusion detection systems* (NIDS) are increasingly becoming critical security components. From a research standpoint, although investigated for many years [16], NIDSs continue to get a lot of attention due to their practical interest and to challenges such as the need to handle large data volumes [17, 27].

Anomaly-based network intrusion detection is a particularly promising approach as it allows detecting previously unknown attacks. This approach resorts to machine learning to create models of normal behavior, then detecting deviations from this behavior. Most of the research in this area uses standard predictors known for decades, such as naive Bayes, thus supervised machine learning.

Deep learning is currently a hot topic in machine learning [3, 4]. This kind of techniques have been achieving excellent results in the recognition of speech, faces, and images, to name a few [3]. Schmidhuber has a recent survey on major work in the area [21].

The term *deep learning* first appeared in the late 2000s associated to recent work in *neural networks* with many layers, or *deep belief networks*. However, with time its semantics expanded to a variety of other methods, as neural networks are equivalent to other statistical and machine learning approaches in structural terms. Recursive generalized linear models have the same structure as deep belief networks. Gradient boosting machines leverage performance via boosting with an ensemble of weak prediction models, typically decision trees.

This paper explores the use of these three deep learning techniques in the context of anomaly-based network intrusion detection: *generalized linear models* (GLM), *gradient boosting machines* (GBM), and *deep learning* (DL), using H2O's denominations [1]. We executed both standard predictors and these advanced techniques with a recent, carefully crafted, data set of network traffic containing known attacks. The goal is to understand if deep learning algorithms have superior performance over standard predictors and at what cost.

The data set is composed of sequences of records that represent traffic flows. Each record is labelled as normal or attack (i.e., as being an attack or part of an attack). In terms of machine learning, the intrusion detection problem can be modelled as a linear regression problem. This means that we use a machine learning predictor (e.g., GBM) configured with the train data set to predict if every record of the test data set is normal or attack. We assess the quality of the prediction using standard metrics such as the mean square error.

The main contributions of this paper are showing that it is possible to discern, in a transverse manner, abnormal inputs from normal in a given set of data by means of the algorithms employed and, more importantly, that new methods very little explored in the literature can convey good results.

The experimental results show that the deep learning algorithms provide a better fit than older algorithms, taking advantage of their superior modelling capacities. As an example the capacity to fit tweedie type distributions has proven to be rather beneficial with the data set we used, since most of the features of the NSL-KDD has tweedie like distributions.

2 Method

The paper presents an experimental comparison of some deep mining algorithms with more classical mechanisms in the context of network intrusion detection. The deep mining algorithms selected were those more popular today and with available implementations. For the classical algorithms there was a vast choice, so we selected a few of different types among those known to provide better results.

The experimental comparison method used was:

– select an adequate data set of network traffic containing attacks;
– tune the detection algorithms;
– run the algorithms with the data set;
– analyse the results.

We start with the first and leave the rest for Section 3.

2.1 The Dataset

Data sets have been widely adopted as a means to benchmark NIDSs (or IDSs in general). Two of the first data sets used for intrusion detection in this scope were DARPA1998 and DARPA1999, developed under the auspices of DARPA. These data sets have been widely used for several years [14, 6]. However, they were also much criticized due to a number of reasons like: they were synthetic, so could lead to overoptimistic results; the attack characterization was not realistic; they did not allow false alarm evaluation [13, 14]. The KDD99 data set was developed in an attempt to improve these two data sets. The definition of this data set involved identifying parameters that would allow effective detection, while minimizing processing costs. The objective was to allow real-time detection. Extensive work has been done over this data set, but many criticisms were also made [25]. However, an extensive study of the KDD99 data set argues that this data set is still valid within the following scope of objectives [9]: dealing with high dimensional data; learning from a very large imbalanced data set; feature selection research; detecting new intrusions and simulating intrusion detection of encrypted data.

The NSL-KDD data set. With such a scarcity of adequate data sets, researchers from the University of New Brunswick (UNB) created a new data set known as the NSL-KDD data set or NSL-KDD data set for short [23]. The NSL-KDD data set is fairly recent and aims at solving, among others, the most hindering problems identified by Travalle et al. in KDD99 [25]. The authors explain that their data set may still suffer from some of the problems discussed by McHugh [14]. McHugh claims that a data set should reflect as much as possible live data, at risk of bias towards unrealistic results with respect to true detection, false alarms, or both. However, the UNB researchers claim that processed data sets have benefits in terms of redundancy reduction in both the train and test sets. Also unbalance issues of the KDD99 data set have been minimized by redistributing the sample proportions of classes. The cardinality of the records corresponding to each difficulty level group is inversely proportional to the percentage of records in the original KDD99 data set. As a consequence, the prediction rates of different machine learning methods vary in a wider range arguably allowing a more precise assessment of distinct learning techniques. In this regard, the NSL-KDD aims at a greater degree of balancing.

The NSL-KDD data set is composed of sequences of entries or records. Each entry represents a *flow*, i.e., a sequence of IP packets starting at a time instant and ending at another, between which data flows to and from a source IP address to a target IP address using a transport-layer protocol (TCP, UDP) and an application-layer protocol (HTTP, SMTP, SSH, IMAP, POP3, or FTP). Each flow is labeled as either *normal* or *attack*. In the version of the data set used, no specifics on the attack types is included, although there were several types of attacks. The attacks fall into four main classes:

- Probing – scan based attacks, e.g., network scans; port sweep;
- Denial of Service (DoS) – network and systems denial of service, e.g., syn flood;
- Remote to Local (R2L) – unauthorized access from a remote machine, e.g., password guess attacks;

- User to Remote (U2R) – unauthorized access to local superuser (root) privileges, e.g., buffer overflow attack on a program running as root.

The data set is composed of two sub-data sets: *train data set*, used for training a NIDS, and *test data set*, used for testing the NIDS. Both have the same structure and contain attacks of the four classes. It is important to note that the test data does not have the same probability distribution as the training data, and it includes specific attack types not in the training data, hence with specific distributions. A total of 37 types of attack are included in contrast with only 23 in the train data set. Table 1 summarizes this information. Each record of the data set is characterized by features that fall into three categories: basic, features, and traffic features. These features are represented in Table 2.

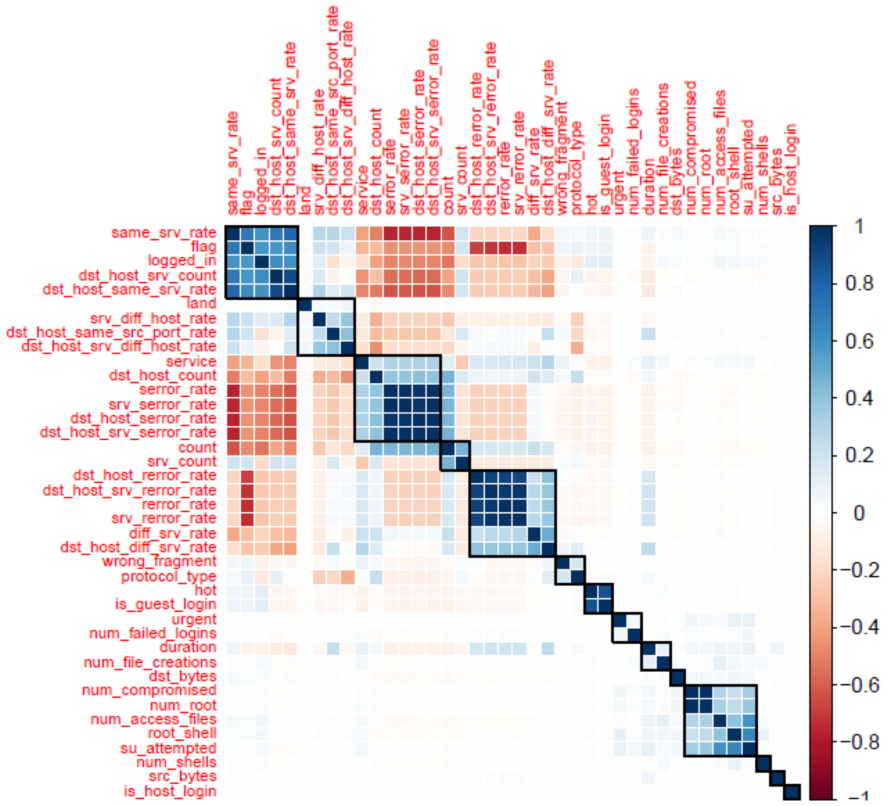

Fig. 1. Correlation Plot of the features included in the NSL-KDD Dataset.

Class	Train data set attacks	Attacks in test data set only
Probe	portsweep, ipsweep, satan, guesspasswd, spy	snmpguess, saint, mscan
DoS	smurf, neptune, land, pod, teardrop, buffer overflow, warezclient, warezmaster	worm, udpstorm
R2L	imap, phf, multihop	snmpget, httptunnel
U2R	perl, loadmodule, ftp write, rootkit	sqlattack , mailbomb, processtable

Table 1. Attacks in the NSL-KDD train and test data sets (all attacks from the train exist also in the test data set)

Feature	Description
duration	length of the flow in seconds
protocol-type	type of the protocol, e.g., TCP, UDP, ICMP
service	network service, e.g., HTTP, telnet
src-bytes	num. of data bytes from source to destination
dst-bytes	num. of data bytes from destination to source
flag	status of the flow, normal or error
lang	1 if flow is from/to the same host/port; 0 otherwise
wrong-fragment	num. of erroneous fragments
urgent	num. of urgent packets
hot	num. of hot indicators
num-failed-logins	num. of failed login attempts
logged-in	1 if successfully logged in; 0 otherwise
num-compromised	num. of compromised conditions
root-shell	1 if root shell is obtained; 0 otherwise
su-attempted	1 if su root command attempted; 0 otherwise
num-root	num. of root accesses
num-file-creations	num. of file creation operations
num-shells	num. of shell prompt
num-access-files	num. of operations on access control files
num-outbound-cmds	num. of outbound commands in an ftp session
is-hot-login	1 if the login belongs to the hot list; 0 otherwise
is-guest-login	1 if the login is a guest login; 0 otherwise
count	num. of connections to the same host as current connection
serror-rate	perc. of connections that have SYN errors
rerror-rate	perc. of connections that have REJ errors
same-srv-rate	perc. of connections to the same service
diff-srv-rate	perc. of connections to different services
srv-count	num. of connections to the same service as current connection
srv-serror-rate	perc. of connections that have SYN errors
srv-rerror-rate	perc. of connections that have REJ errors
srv-diff-host-rate	perc. of connections to different hosts

Table 2. Features used to characterize each flow in the data set: basic (top), content (middle), traffic (bottom).

2.2 Tools

All experiments were performed with R which is a programming language and software environment for statistical computing and graphics supported by the R Foundation for Statistical Computing. The R language is widely used among statisticians and data miners for developing statistical software and data analysis [19]. It is a GNU project that provides a wide variety of statistical (linear and nonlinear modeling, classical statistical tests, time-series analysis, KNN, clustering etc.) and graphical techniques, and is highly extensible. R provides tools for data pre-processing, classification, regression, clustering, association rules, and visualization.

The R packages H2O [1], kknn [20], rpart [26] and e1071 [15] were used in the analysis process. The main package was H2O, that implements deep learning algorithms for R. It calls the H2O software package (Apache 2.0 License) produced by H2O.ai (previously 0xdata). The H2O R package provides a number of implementations of machine learning algorithms: GLM, GBM, deep learning (multi-layer neural network models), K-means, naive Bayes, principal components analysis, principal components regression, and random forest.

2.3 Performance metrics

To assess our results we use a set of metrics adequate for evaluating regression models: Mean Square Error, Root Mean Square Error, R Squared (also named Coefficient of Determination). The commonly used error indicator is not used since its not the most adequate for regression models.

Mean Square Error. The mean squared error (MSE) of an estimator measures the average of the squares of the errors or deviations between the estimated values ($\hat{y}_i$) and the real values (y_i). In other words, it gives the average of the squared differences between the estimator and what is estimated: $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

Root Mean Square Error. The square root of MSE. The use of RMSE is very common as it makes an excellent general purpose error metric for numerical predictions. Compared to the similar mean absolute error, RMSE amplifies and severely punishes large errors. Root Mean Square Error is given by $RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$

Root Square Error. The square root of the square of the errors. It provides a measure of error without error signal bias. $RSE = \sqrt{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$

R Square. R Square, also called coefficient of determination, R^2 is the proportion of variability in a data set that is accounted for by a statistical model. In other terms it is an indicator of how well data fits a statistical model. In this definition, the term variability is defined as the sum of squares: $R^2 = 1 - \frac{SS_{res}}{SS_{tot}} = 1 - \frac{\sum_i (y_i - \bar{y})^2}{\sum_i (f_i - \bar{y})^2}$ The values of f_i are the predicted values in the model.

3 Results

This section presents the study itself. The section starts with the older algorithms (Section 3.1), then presents the advanced, deep learning, schemes (Section 3.2). For each algorithm we explain how it works, then explain how we tuned its parameters in order to obtain results as good as possible, then present its results for intrusion detection.

3.1 Classical algorithms

This section explores three algorithms: k-nearest neighbors, decision trees, and support vector machines. These were chosen as representative of some of the most tested algorithms in the realm of NIDS.

k-Nearest Neighbors. The k-Nearest Neighbors algorithm (KNN) is a non-parametric method used for classification and regression. In both cases, the classes are grouped according with the k closest training examples in the feature space. KNN can be used for classification or regression. Here we use it for regression.

A test data set, different from the training data set but with the same shape and features as the training data has been used for model validation (i.e., production of error rates on data not used in model building). A value of maximum value for k of 15 has been chosen from which an optimal value of 5 was derived. A number of kernels has been tested from which the triangular kernel was chosen as best.

The algorithm has been tested against a set of possible kernel types, namely triangular, rectangular and epanechnikov, and optimal with kmax of 15 and distance of 3. Kernel methods introduce non-linearities through implicit non-linear or linear transforms, often local in nature. The main objective is to facilitate separability, i.e. the closer two points are, the larger the kernel value. In general terms and therefore in the KNN algorithm, a kernel transformation of the mentioned types takes place before the model is processed.

Experimental results. The results for the KNN algorithm optimization phase run over the training data are presented in Table 3. From this optimization test, a triangular kernel with k=6 has been run with the results depicted in Table 4.

Mean Error	MSE	Best kernel	Best k
0.003	0.003	triangular	6

Table 3. Results for KNN algorithm optimization phase run over the training data

MSE	RMSE	RSE	R Square
0.228	0.447	0.929	0.071

Table 4. Results for KNN

Decision Trees. Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. The deeper the tree, the more complex the decision rules and the fitter the model.

Experimental results. An rpart algorithm has been tested against the training set. Prediction values have been extracted with the test set. The results are in Table 5.

Support Vector Machines. SVMs are a group of classifiers usually considered to be one of the best out of the box methods. There are several types of SVMs. The simplest is

MSE	RMSE	RSE	R Square
0.200	0.447	0.817	0.094

Table 5. Results for Rpart decision tree

the maximal margin classifier (MMC). Though the MMC is elegant and simple, it cannot be applied to most data sets, since it requires the classes to be separable by a linear boundary. The support vector classifier is an extension of the maximal margin classifier, and can be applied in a broader range of cases. However, better yet, the support vector machine is a further extension to accommodate non-linear class boundaries.

SVM tuning Using the tune.svm primitive of the e1071 R package, a tuning phase took place in order to optimize SVM performance over the choice of the best choices for the Cost and Gamma values, sweeping a range of values for gamma from 10^-6 to 10^-3 and the value for Cost from 10^1 to 10^3. Optimal values for Cost around 1000 and for gamma around 0.0001 have been found (Fig. 2).

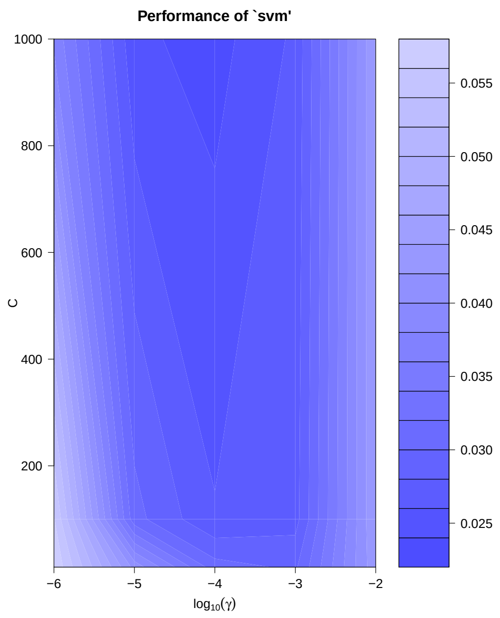

Fig. 2. SVM optimization graph with gradient values for Gamma and Cost

Experimental results. SVM exhibited a strong trade-off between the number of support vectors and the training error with a marked difficulty in this tuning process if not assisted by a tuning phase. Optimal values of C=1000 and gamma=0.0001 have been used. In the training phase it has become evident, in particular for the optimized parameters used, the extreme inefficiency of the SVM algorithm as the training time was measured in hours, incomparably longer than in the comparative cases. Results in Table 6.

MSE	RMSE	RSE	R Square
0.200	0.447	0.817	0.0948

Table 6. Results for SVM

3.2 Advanced algorithms

Improved predictive power has allowed Deep Learning to be considered as a valuable machine learning technique. Unlike the neural networks of the past, modern Deep Learning has improved training stability, generalization and scale on big data and is quickly becoming the algorithm of choice for high-level predictive accuracy.

The most popular deep learning algorithms are Deep Boltzmann Machine (DBM), Deep Belief Networks (DBN), Convolutional Neural Network (CNN) and Stacked Auto-Encoders.

Generalized Linear Models. Generalized linear models (GLM) are a generalization of linear regressions. Linear regression models the dependency of response y on a vector of predictors $x(y \sim x^T\beta + \beta - 0)$. The models are built with the assumptions that y has a Gaussian distribution with a variance of σ^2 and the mean is a linear function of x with an offset of some constant $beta - 0$, i.e. $y = \mathcal{N}(x^T\beta + \beta - 0; \sigma^2)$. These assumptions can be overly restrictive for real-world data that do not necessarily follow have a Gaussian distribution. GLM generalizes linear regression in the following ways:

- adds a non-linear link function that transforms the expectation of response variable, so that $link\,(y) = x^T\beta + \beta - 0$
- allows variance to depend on the predicted value by specifying the conditional distribution of the response variable or the family argument.

Experimental results. Results extracted with the h2o.glm function of the H2O package can be found in Table 7. Performance indicators have been extracted with the training data and with a different test data.

Gradient Boosting Machines. Gradient Boosting Machines (GBM) is an algorithm designed to produce a prediction model in the form of an ensemble of weak prediction models. It encloses the concepts of Gradient Descent and Boosting; hence the name. It builds the model in a stage-wise fashion and is generalized by allowing an arbitrarily differentiable loss function. GBM fits consecutive trees where each solves for the net error of the prior. The idea of gradient boosting originated in the observation by Breiman [5] that boosting can be interpreted as an optimization algorithm on a suitable cost function.

Performance metrics	Results with training data	Results with test data
MSE	0.040	0.038
R Square	0.840	0.844
Mean Residual Deviance	0.040	0.039

Table 7. Results for GLM

Experimental results. Results extracted with the h2o.gbm function of the H2O package are in Table 8. Number of folds for cross-validation: nfolds=10. Performance indicators have been extracted with the training data and with a different test data.

Performance metrics	Results with training data	Performance results with test data
MSE	0.041	0.169
R Square	0.836	0.310
Mean Residual Deviance	0.041	0.170

Table 8. Results for GBM

Deep Learning. H2O Deep Learning in unsupervised mode currently only supports (deep) autoencoding, where all layers are trained at once, just like a regular feed-forward neural net with features output neurons and using the MSE loss function to attempt to train the identify function.

Unsupervised pre-training with RBMs to build a DBN is only required when you have insufficient labeled data, but H2O's Deep Learning was really designed for cases where there are sufficient labels, similar to the other classifiers, such as RF/GBM/GLM, so it performs best in fully supervised mode.

Experimental results. Results extracted with the h2o.deeplearning function of the h20 package are in Table 9. Performance indicators have been extracted with the training data and with a different test data.

Performance metrics	Results with training data	Results with test data
MSE	0.041	0.169
R Square	0.836	0.310
Mean Residual Deviance	0.041	0.169

Table 9. Results for Deep Learning

4 Analysis and Discussion

We summarize the experimental results in two tables: KNN, Decision Trees and Support Vector Machines in Table 10, Deep Learning, GLM and GBM in Table 11. All results are reproducible without need to establish seed values due to the algorithms stochastic convergent behavior.

Predictive accuracy of the algorithms under study are the Error, the Root Mean Squared Error (RMSE), Mean Square Error (MSE), Root Mean Square Error (RMSE), Root Squared Error(RSE) and R Square (R2). Another metric applied to deep learning is the Mean residual Deviance (MRD). All results reported are for the training set and test set.

The predictive estimation results are summarized in Table 10 for KNN, Decision Trees and SVMs. There is a obvious confluence of performance indicators among the three types of algorithms. All the three show rather close figures for all the performance indicators extracted. According to what could be expected all the three extract a model that allows rather close predictions for the test data. In itself this is a good behavior that justifies using the same algorithms comparatively.

Regression Metrics	KNN	Decision Tree	SVMs
Mean Square Error	0.230	0.200	0.222
Root Mean Square Error	0.480	0.447	0.471
Root Square Error	0.940	0.817	0.906
R Square	0.186	0.094	0.094

Table 10. Summary of results for KNN, Decision Trees and Support Vector Machines

The predictive estimation results are summarized in Table 11 for GBM and Deep Learning. The results are clearly better than the other table, with errors and higher R Square. When comparing the predictive results for GBM and Deep Learning, it is noticeable a degree of overfitting when the performance metrics are extracted over the training data. This is expectable as the model is built on the same data.

Regression Metrics	DL-Train	DL-Test	GLM-Train	GLM-Test	GBM-Train	GBM-Test
Mean Square Error	0.041	0.169	0.040	0.040	0.005	0.168
R Square	0.836	0.310	0.840	0.844	0.979	0.316
Mean Residual Deviance	0.041	0.169	0.040	0.040	0.005	0.168

Table 11. Summary of results for the advanced algorithms tested

5 Related Work

A number of anomaly detection techniques related with malicious activity have been studied in the fields of computing, communications, and networking [18]. Some examples are: anomaly detection in data in computers and systems, anomaly detection in network traffic, detection of network configuration and vulnerabilities, assessing network and data integrity, recognition of attack patterns, tracking policy violations, and analysis of abnormal activities in logs. A particularly exhaustive review has been done by Shan et al. [22].

Anomaly detection systems were first described by Denning in her seminal work on the host-based intrusion-detection expert system (IDES) that described a model of a real-time expert system capable of detecting break-ins, penetrations, and other forms of breach [8]. Denning's model describes profiles for representing the behavior of subjects with respect to objects in terms of metrics and statistical models, and rules for acquiring knowledge about this behavior from audit records and for detecting anomalous behavior. This model lay the ground on modeling data sets identifying anomalies to be used for machine learning intrusion detection systems (IDS), NIDSs included. Subsequent machine learning approaches have been tested against attacks and intrusions of several types, e.g., attacks against web servers [12].

Deepa and Kavitha [7] surveyed a comprehensive set of machine learning approaches for intrusion detection, namely association-rules, data clustering, Bayesian networks, hidden Markov models, decision trees, support vector machines [11], genetic algorithms and honey pots. Garcia-Teodoro et al. [10] have produced another survey of anomaly-based intrusion detection techniques, outlining the main challenges facing wide scale deployment of anomaly-based intrusion detectors, with special emphasis on assessment issues. Sommer and Paxson [24] have noted a gap between the amount of research papers and the number of implementations of IDSs based on machine learning and identified several difficulties.

There is some preliminary work on applying deep learning to network intrusion detection that achieved a detection accuracy of about 0.975 [2].

6 Conclusions

From the comparative analysis of the various algorithms tested the following conclusions can be inferred.

The set of “classical" algorithms proves to convey equivalent results when the same test conditions are essayed. This in itself supports the perspective conveyed by by Engen at al. [9] regarding the quality of the NSL-KDD data set as successor of the KDD99. We recall that one of the main objections regarding the value attributed to the KDD99 data set was related to the disparity of results obtained. In this regard the results obtained for the NSL-KDD data set seem to indicate a sustained similitude of results given the same experimental conditions.

Again in itself this is a convenient property to use the given algorithms in an interchangeable way for instance in ensemble arrangements. An over fitting phenomena can be observed in GBM and Deep learning when testing with training data. This is a common property regarding learning algorithms pointing that in this particular case and for algorithm evaluation purposes the best approach is to test the models obtained with a distinct test set. Given the similar values for R2 and MSE obtained for GBM and Deep Learning tested against the test set we may assume a good degree of similitude for prediction between these two algorithms. Taken at face value the results seem to indicate good adaptability for ensemble regression.

When compared with the classical cases we may notice a better performance judging from the values of MSE and R2 obtained: MSE is consistently lower for MSE and Deep Learning and the bigger values of R2 obtained seem to indicate that the models do a better job at assimilating the variance of the train data.

Acknowledgements. This research was supported by by the European Commission under grant agreement number 830892 (SPARTA) and by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UIDB/ 50021/2020 (INESC-ID).

References

1. S. Aiello, T. Kraljevic, and P. Maj. *h2o: R Interface for H2O*, 2015. R package version 3.6.0.8.
2. M. Z. Alom, V. Bontupalli, and T. M. Taha. Intrusion detection using deep belief networks. In *Proceedings of the 2015 National Aerospace and Electronics Conference*, pages 339–344, June 2015.
3. G. Anthes. Deep learning comes of age. *Communications of the ACM*, 56(6):13–15, 2013.
4. I. Arel, D. C. Rose, and T. P. Karnowski. Deep machine learning: A new frontier in artificial intelligence research. *IEEE Computational Intelligence Magazine*, 5(4):13–18, 2010.
5. L. Breiman. Arcing the Edge. Technical report, Statistics Department, University of California, Berkeley, June 1997.
6. C. Brown, A. Cowperthwaite, A. Hijazi, and A. Somayaji. Analysis of the 1999 DARPA/Lincoln laboratory IDS evaluation data with NetADHICT. In *Proceedings of the 2nd IEEE International Conference on Computational Intelligence for Security and Defense Applications*, pages 67–73, 2009.
7. A. Deepa and V. Kavitha. A comprehensive survey on approaches to intrusion detection system. *Procedia Engineering*, 38:2063–2069, 2012.
8. D. E. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, 13(2):222–232, Feb. 1987.
9. V. Engen, J. Vincent, and K. Phalp. Exploring discrepancies in findings obtained with the KDD Cup'99 data set. *Intelligent Data Analysis*, 15(2):251–276, Apr. 2011.
10. P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security*, 28(1-2):18–28, Feb. 2009.
11. L. Khan, M. Awad, and B. Thuraisingham. A new intrusion detection system using support vector machines and hierarchical clustering. *The VLDB Journal*, 16(4):507–521, Oct. 2007.
12. C. Kruegel and G. Vigna. Anomaly detection of web-based attacks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, 2003.
13. M. V. Mahoney and P. K. Chan. An analysis of the 1999 DARPA/Lincoln laboratory evaluation data for network anomaly detection. In *Proceedings of the 6th International Symposium on Recent Advances in Intrusion Detection*, pages 220–237. Springer, 2003.

14. J. McHugh. Testing intrusion detection systems: A critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Transactions on Information and System Security*, 3(4):262–294, Nov. 2000.
15. D. Meyer, E. Dimitriadou, K. Hornik, A. Weingessel, and F. Leisch. e1071: Misc functions of the department of statistics, probability theory group, 2015. R package version 1.6-7.
16. B. Mukherjee, L. T. Heberlein, and K. N. Levitt. Network intrusion detection. *IEEE Network*, 8(3):26–41, 1994.
17. H. A. Nguyen and D. Choi. Application of data mining to network intrusion detection: Classifier selection model. In *Challenges for Next Generation Network Operations and Service Management, 11th Asia-Pacific Network Operations and Management Symposium*, pages 399–408, 2008.
18. A.-S. K. Pathan, editor. *The State of the Art in Intrusion Prevention and Detection*. Auerbach Publications, 2014.
19. R Foundation. R Project, 2015. https://www.r-project.org/about.html.
20. K. Schliep and K. Hechenbichler. *kknn: Weighted k-Nearest Neighbors*, 2015. R package version 1.3.0.
21. J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.
22. A. A. Shah, M. S. H. Khiyal, and M. D. Awan. Analysis of machine learning techniques for intrusion detection system: A review. *International Journal of Computer Applications*, 119(3):19–29, June 2015.
23. A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers & Security*, 31(3):357–374, May 2012.
24. R. Sommer and V. Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, pages 305–316, 2010.
25. M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani. A detailed analysis of the KDD CUP 99 data set. In *Proceedings of the 2nd IEEE International Conference on Computational Intelligence for Security and Defense Applications*, pages 53–58, 2009.
26. T. Therneau, B. Atkinson, and B. Ripley. rpart: Recursive partitioning and regression trees, 2015. R package version 4.1-10.
27. T.-F. Yen, A. Oprea, K. Onarlioglu, T. Leetham, W. Robertson, A. Juels, and E. Kirda. Beehive: large-scale log analysis for detecting suspicious activity in enterprise networks. In *Proceedings of the 29th ACM Annual Computer Security Applications Conference*, 2013.