Machine Learning Techniques for Intrusion Detection

Yih Han Tan

Machine Learning Techniques for Intrusion Detection

GIAC (GCIA) Gold Certification

Author: Yih Han Tan, tanyihhan@gmail.com
Advisor: Hamed Khiabani, Ph.D.
Accepted: June 2nd 2021

Abstract

This paper aims to equip intrusion analysts with the basic techniques needed to apply machine learning to intrusion detection. It will first review and describe the different approaches to machine learning-based classification (e.g., logistic regression, support vector machines) before explaining the challenges of applying it to network intrusion detection. It will also review methods of data preprocessing, model training, and testing. This paper then describes experiments carried out on a dataset (NSL-KDD) that is widely used to test intrusion detection algorithms. Two sets of experiments demonstrating the application of commonly used machine learning-based classification and methods extensively used to improve model performance (e.g., boosting, bagging, stacking, label smoothing, and embedding) are performed. With a knowledge of the underlying algorithms and the provided source code, network operators can experiment with and eventually apply machine learning-based intrusion detection to their network.

1. Introduction

Network intrusion detection systems (NIDS) can be broadly classified into two categories, based on the method of detection: misuse detection and anomaly detection. Systems using misuse detection rely on precise indicators (signatures) of known malicious behavior, while anomaly detection systems rely on a baseline of regular activities and flag any deviations from this set of activities. The techniques described in this paper will help the development of an anomaly detection system, which can be viewed as a classification system, classifying network traffic as anomalous or not.

Such a classification system requires a model that is trained by "machine-learning" algorithms (Gron, 2017). The first step of model training is feature selection. Features are characteristics in the data that could help the model decide. For example, a pigeon/eagle classification model that identifies whether a previously unseen bird is a pigeon or an eagle will likely use characteristics like wingspan or length of its beak.

After identifying the features, the model must be trained with example data (the training set). Each training example is a training instance. For this example, each training instance comprises measurements of the bird's wingspan and beak length and a label (whether the data were collected from a pigeon or an eagle).

The training process will help the model determine whether a previously unseen bird is a pigeon or an eagle. One possible method is to learn a threshold equidistant to the shortest wingspan measured on eagles and the longest wingspan measured on pigeons in the training set. When classifying a new test sample, the unknown bird will be classified as a pigeon if the wingspan is shorter than the threshold or as an eagle, if otherwise. This is the main idea behind a class of classification algorithms called support vector machines which will be discussed later.

Another example closer to home is a spam filter that can flag spam emails after being given examples of both spam emails and regular emails. The classification model behind a spam filter typically comprises features based on the occurrence and distributions of words and phrases. Recently discovered techniques that boosted the effectiveness of neural net-

based language models have greatly improved the performance of tasks that require natural language processing. These tasks include machine language translation, text search, and text classification. These techniques are also driving the performance improvement of spam filters.

Machine learning algorithms can be best classified according to the amount of supervision they receive during training. There are four major classes: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. What was listed earlier are examples of supervised learning, more specifically, classification. The rest of this paper will describe the application of supervised learning on intrusion detection in greater detail.

1.1. Application of Machine Learning on Intrusion Detection

Most commercial intrusion detection systems are signature-based. These systems compare incoming traffic with an existing database of known attack patterns known as signatures. Vendors of these systems frequently release new signatures as they received information on new attack vectors. As they depend on signature matches, it is unlikely that a signature-based system can pick up malicious traffic from previously unknown attack vectors. Since an anomaly detection-based intrusion detection system uses a model trained on both malicious and non-malicious traffic, it can learn common characteristics of both classes and potentially pick up malicious traffic from previously unseen attack vectors.

The application of machine learning to improve intrusion detection is an active area of research (Sinclair, Pierce, S. Matzner, 1999) (Mayhew, Atighetchi, Adler, Greenstadt, 2015). Researchers have applied a wide range of machine learning methods to intrusion detection. These include both traditional methods like logistic regression (Shah, 2017), SVM (Goeschel, 2016) and clustering (Peng, 2018) and neural network-based methods (Long, 2018) (Liu, 2019). In a related application of machine learning, the work in (Rigaki, 2018) used recently discovered generative adversarial networks (GAN) to modify malware-generated network traffic to ensure that they are similar to the traffic of a legitimate application to avoid detection.

Detection can be either flow-based or packet-based. Flow-based detection operates on flow data containing packets grouped into periods. Detecting attacks with flow data is beneficial as flows represent the whole network environment, enabling the detection of most attacks, and flow-based detection can be fast and straightforward. However, since packet content is not inspected, detection performance for certain attacks (e.g., user to root (U2R), remote to local (R2L) may be sub-optimal. Feature engineering is an essential step before model training. The common features include the average packet length, the variance in packet length, and the duration of connections.

Packet-based detection operates on packets, the basic network communication units, containing details of each communication. A packet consists of a header and application data. The headers are structured fields that specify IP addresses, ports, and other fields specific to various protocols. The application data portion contains the payload from the application layer protocols. Using packets as IDS data sources is advantageous as packets contain 1) communication contents, so they can be used to detect U2R and R2L attacks, and 2) IPs and timestamps, so they can be used to locate the attack sources precisely. However, individual packets do not reflect the whole communication state nor the contextual information of each packet, so it is difficult to detect some attacks. Since the packet content, which may contain sensitive information, is required for model training, finding the necessary training data will be even more challenging.

1.2. Limitations of Machine Learning

Machine learning has been applied to many areas commercially (e.g., machine language translation, facial recognition, and product recommendation), leading to significant performance improvements. Though the potential benefits of applying machine learning to intrusion detection are apparent, there are difficulties that must be overcome before similar performance improvements can be realised (Sommer, Paxson, 2010).

Applying machine learning to intrusion detection is often hampered by a lack of training data. This is mainly due to the data's sensitive nature. Network data may contain confidential communications or personal information, making it difficult and risky for

researchers to share them with the research community. As a result, many datasets used by the community are obtained through simulation.

Though the KDD99 (KDD99, 1999) and other related datasets are the most widely used, they are problematic (McHugh, 2000):

1) The data are severely unbalanced, making the classification results biased toward the majority classes.
2) There are many duplicate records that must be filtered. As a result, the experimental results from different studies are not always comparable.
3) KDD data do not represent the current network environment.

During the selections of records from KDD99 to create the NSL-KDD, there were efforts to balance the records of different classes and remove duplicates. Therefore, the experiments can be implemented on the whole dataset, and the results from different papers are consistent and comparable. While the NSL-KDD reduces the problems of data bias and data redundancy to some degree, it does not contain new data and remains outdated.

Although machine learning methods can detect intrusions, they typically do not perform well on data that all too different from the training data. Consequently, when the training dataset does not cover all typical traffic, good performance during operation may not be possible, even if the models achieve high accuracy on test sets. Network traffic often exhibits much more diversity than expected, even within a single network, the network's characteristics (e.g., bandwidth, duration of connections, and applications) can vary greatly over time.

Furthermore, in intrusion detection, the relative cost of any misclassification is extremely high compared to many other machine learning applications (e.g., machine language translation or product recommendation). False positives waste an analyst's time and can result in an NIDS being unusable. False negatives, on the other hand, could cause serious damage to an organization.

Though these challenging problems are barriers to the successful application of machine learning to intrusion detection, there is also an opportunity for network operators

to add immense value to the defense of their network by collecting data and training intrusion detection models that are adapted to their network. This is especially valuable as the lack of good open-source training data means that it will be difficult for vendors to provide a good solution without intimate knowledge of the network the IDS will be deployed in.

2. Machine Learning Models

This section describes a few commonly used machine learning models. While a deep understanding of what happens under the hood for different models is often unnecessary when applying them to various problems. Understanding of the basics of each algorithm is useful during model selection and parameter adjustment to improve model performance.

2.1. Logistic Regression

Logistic regression is commonly used to estimate the probability that a test sample belongs to a particular class. Logistic regression applies a sigmoid function to the output of the linear regression. During classification, if the estimated probability is greater than 50%, then the model predicts that the instance belongs to that class (called the positive class, labeled "1"), or else it predicts that it does not (i.e., it belongs to the negative class, labeled "0").

A linear regression model is typically of the form: $z = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_N x_N$, while a logistic regression model is of the form: $h(z) = \frac{1}{1+e^{-z}}$, where $\theta_0, \ldots, \theta_N$ are model parameters that will be updated during training and $x, \ldots, x_N$ are value of features of the test sample.

2.1.1. Model Training

The cost function used for logistic regression is the average of the log loss across all training examples:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)}\log(h(z(\theta)^{(i)})) + (1 - y^{(i)})\log(1 - h(z(\theta)^{(i)})),$$

where m is the number of training samples, $y^{(i)}$is the actual label of the i[th] training sample and $h\left(z(\theta)^{(i)}\right)$ is the model's prediction for the i[th] training example.

The loss function for a single training example is:

$$J(\theta) = -\left[y^{(i)} \log\left(h\left(z(\theta)^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - h\left(z(\theta)^{(i)}\right)\right)\right].$$

It can be observed that when the model makes a correct prediction for a training sample, the loss is close to 0. Otherwise, the loss is close to 1.

Logistic regression is useful as it is easy to implement and interpret. The model coefficients can be interpreted as indicators of feature importance. It is also reasonably accurate for simple datasets where different classes are linearly separable. Since the logistic regression function does not capture more complex relationships between the features, it will underperform algorithms like support vector machines and neural networks when applied to more complex datasets.

2.2. Support Vector Machine

The training of support vector machines involves finding the optimal hyperplane for two linearly separable classes of training samples. Support vectors are the training samples that lie closest to the decision hyperplane, i.e., the optimal hyperplane will change if any of them is removed. The distance between the hyperplane and the nearest data point from either set is known as the margin. After obtaining the hyperplane after training, classifying a test sample involves simply determining which side of the hyperplane is the sample.

2.2.1. Model Training

The goal is to choose a hyperplane with the greatest possible margin between the hyperplane and any point within the training set, giving a greater chance of new data being classified correctly.

The basic SVM algorithm has shortcomings when dealing with data that are not linearly separable that could impact performance. The first shortcoming arises when the algorithm strictly places the training data on the right side of the hyperplane based on their

labels. This only works if the samples can be grouped into linearly separable classes and makes the training process sensitive to outliers. Soft margin classification is a technique that can be used to overcome this problem. By allowing margin violations, it enables a more flexible model that could generalise better. The more margin violations permitted during training, the more widely the algorithm can separate the different classes as it does not have to accommodate the samples that are the most different to accommodate when determining the hyperplane. In most implementations, a hyperparameter (c in Scikit-Learn) controls the balance between separating the samples from different classes as widely as possible and limiting the number of margin violations.

While linear SVM classifiers work well in many cases, many datasets are not linearly separable. One approach to overcome this problem is to augment the data by adding more features (e.g., polynomial ones). The augmented dataset may become linearly separable. Unfortunately, features with high polynomial degrees may be required to deal with complex datasets. This could significantly increase the number of features and render the resulting model too challenging to train. A technique called the kernel trick helps overcome this problem. Applying the kernel trick helps achieve the same result as adding many polynomial features without the exponential increase in the number of features.

Techniques like soft margins and kernel tricks make the SVM a highly effective tool, especially suitable for the classification of complex datasets that are not large.

2.3. Decision Trees

Decision tree classifiers use a decision tree to map observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). Tree models where the target variable can take a discrete set of values are called classification trees. The prediction algorithm starts at the root node and moves up the tree, picking branches based on conditions at each node. When it reaches a leaf node, it simply predicts the sample as the class for the node.

Performance of decision can be significantly improved by boosting (e.g., xgBoost) and bagging (e.g., random forest). These algorithms will be discussed in greater detail in the section on improving model performance.

2.3.1. Model Training

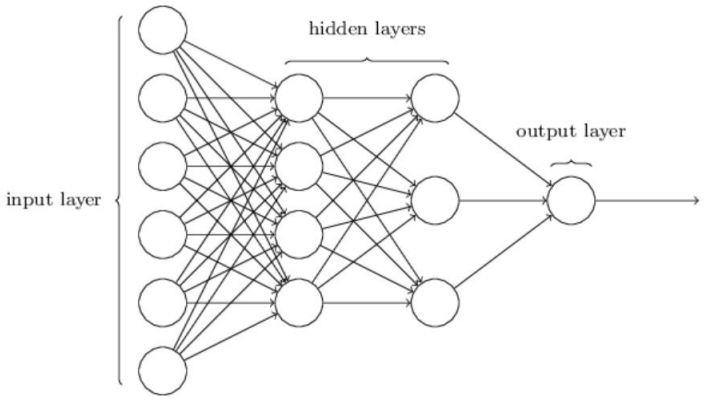
Algorithms for constructing decision trees usually work top-down by choosing a variable at each step that best splits the training dataset. Different algorithms use different metrics for measuring the quality of splits. These typically measure the homogeneity of the target variable within the subsets. These metrics are applied to each candidate subset, and the resulting values are combined (e.g., averaged) to provide a measure of the quality of the split. One example of these metrics is the Gini impurity, a measure of how often a randomly chosen sample from the training set would be incorrectly labeled if it were randomly labeled according to the distribution of labels in the subset. The Gini impurity can be computed by summing the probability p_i of an item with label i being chosen times the probability $\sum_{k \neq i} p_k = 1 - p_i$ of a mistake in categorising that item. It reaches its minimum (zero) when all cases in the node fall into a single target category.

Decision trees are versatile machine learning algorithms that perform well on both classification and regression tasks. They are powerful algorithms capable of fitting complex datasets. One of the best qualities of decision trees is that they require little data preparation. They do not require any feature scaling or centering.

Yih Han Tan, tanyihhan@gmail.com

2.4. Neural Network

A neural network consists of connected nodes, each node holds a number, and each connection holds a weight. Nodes are organised into three layers: input, hidden, and output.



Suppose the network has been trained, the input features of a sample are fed to the input layer. Each node will manipulate its input according to an activation function and an offset. The output will then be fed into the nodes in the next layer. Every link between 2 nodes is assigned a different weight.

During prediction (i.e., a forward pass), the activations of nodes in layer i is:

$$\boldsymbol{a^i} = \sigma(\boldsymbol{W a^{i-1}} + \boldsymbol{b}),$$

where $\boldsymbol{W}$ is a matrix containing the connection weights of each node in layer $i-1$ to nodes in layer i. If layer i has N nodes and layer i-1 has M nodes, W will be N by M. b contains each node's bias and σ is the activation function (e.g., a sigmoid function, or the more commonly used rectified linear unit).

Advances in neural network-related techniques have significantly improved the performance of computer vision and language processing applications. However, neural networks may not outperform decision tree-based models for tabular data. Here, neural network models are built to perform the same classification task for intrusion detection to illustrate how several recently discovered techniques can be used to improve the performance of such models.

Yih Han Tan, tanyihhan@gmail.com

2.4.1. Model Training

The "learning" in neural networks takes place during backpropagation. Backpropagation involves computing the partial derivative ($\partial C/\partial w$) of a cost function C (a measure of similarity between predicted values and the ground truth) with respect to each weight w (or bias b) in the network and adjusting each weight and bias in ways that reduce C.

There are some recently introduced techniques that can help improve the performance of neural networks. Two examples label embedding and label smoothing. Label embedding maps categorical values in an n-dimensional space, enabling models that capture relationships between categories. Label Smoothing involves computing cross-entropy cost with a weighted mixture of actual targets and the uniform distribution. This makes the model less certain of the labels in the training set and often results in less overfitting.

Neural networks' flexibility allows them to learn complex, non-linear relationships, often leading improved prediction or classification performance. However, to learn these relationships, a neural network with many parameters (weights and bias) must be used. A large dataset is required to train a large network effectively.

3. Improving Model Performance

Ensemble learning techniques are used to combine the decisions from multiple models to improve overall performance.

3.1. Boosting

Boosting is an ensemble algorithm that reduces both bias and variance in supervised learning. It works by converting weak learners to strong ones.

AdaBoost is a widely used boosting algorithm. Model fitting starts by training a decision tree, with each observation being assigned the same weight. After evaluating the performance of the first tree, the weights of those observations that are difficult to classify are raised, and those that are easy to classify are lowered.

Another boosting algorithm, gradient boosting, works by successively adding predictors to an ensemble, each one correcting its predecessor. However, instead of adjusting the instance weights at every iteration as AdaBoost does, this method tries to fit the new predictor to the residual errors made by the previous predictor.

3.2. Bagging

One way to improve model performance is to get a diverse set of classifiers using different training algorithms and to consider all their output when deriving the final classification. Another approach to generating multiple classifiers is to use the same training algorithm for all predictors but to train them on different, randomly sampled subsets of the training set. When sampling is performed with replacement, this method is called bagging (short for bootstrap aggregating). The widely used random forest algorithm is an ensemble of decision trees, generally trained via the bagging method.

3.3. Stacking

During stacking, a model is trained to aggregate the predictions of all predictors in an ensemble. Stacking works best when there is a diversity of models, and the predictions made by the different models and the prediction errors made by them are not highly correlated.

4. Tools

Most of the tools essential for applying machine learning (e.g., open-source implementation of model learning, data preprocessing, and performance evaluation algorithms) are widely available.

4.1. Machine Learning Packages

The machine learning packages Scikit-learn and TensorFlow were used extensively for the experiments.

Scikit-learn is a machine learning library for the Python that contains implementation of algorithms for both regression and classification. To achieve high-performance array

and linear algebra operations, Scikit-learn uses NumPy. For further performance improvement, some core algorithms are written in Cython.

Google's TensorFlow is a free and open-source software library for machine learning. Tensorflow is a symbolic math library based on dataflow and differentiable programming.

4.2. Data Preprocessing

Label encoding and data scaling/centering are probably the two most common data preprocessing required before model training.

Many datasets may contain text or categorical values (e.g., days of the week, gender, or colors). Some algorithms like decision trees can handle categorical values, but most of the other algorithms expect numerical values to achieve good performance. There are two main methods to convert categorical values to numerical ones, one-hot encoding and label encoding, both parts of the Scikit-learn library.

Centering is a technique where the mean of independent variables is subtracted from all the values, resulting in all independent variables having zero mean. Scaling divides variables by their standard deviation, resulting in them having a standard deviation of one. Distance-based models (e.g., SVM) perform better when data is centered and scaled.

4.3. Performance Assessment

Accuracy is not an ideal performance measure for classifiers, especially when dealing with datasets that are skewed (i.e., when some classes are much more frequent than others), so more effective performance metrics are required. Precision is the fraction of relevant instances (in this case, malicious traffic) among all flagged instances. The recall is the fraction of flagged instances among all relevant instances.

Precision is defined as $\frac{True\ Positive}{True\ Positive+False\ Positive}$, while recall is defined as $\frac{True\ Positive}{True\ Positive+False\ Negative}$. If precision and recall are of equal importance, the F1-score can be used as a measure of model performance. The F1, which is the harmonic mean of precision and recall is defined as $2 * \frac{Precision*Recall}{Precision+Recall}$.

5. Experiments

The experiments aim to provide a rough gauge of the various algorithms' relative performance and demonstrate how different techniques can be applied to improve model performance. Performance can likely be further improved by hyperparameter tuning and more rigorous cross-validation for each algorithm. That is, to hold back a subset of the training sample and not use it for training. This subset is then used to estimate how the model is expected to perform in general when used to make predictions on data not used during the model training.

The code that generated the results is shared (https://github.com/yihhan/giac.git) for readers to perform further experiments and improve model performance.

5.1. Dataset

The DARPA1998 dataset was curated by the Lincoln laboratory of MIT and is a widely used benchmark dataset in IDS studies. It was compiled from nine weeks of internet data. The training set used data from the first seven weeks, while the test set used data from the last two weeks. The dataset contains both raw packets and labels. There are five types of labels: normal, denial of service (DOS), Probe, User to Root (U2R), and Remote to Local (R2L).

The KDD99 contains 41-dimensional features extracted from DARPA1998. The labels in KDD99 are the same as the DARPA1998. There are four types of features in KDD99, i.e., basic features, content features, host-based statistical features, and time-based statistical features. The records in the NSL-KDD were selected from the KDD99. Records of different classes are more balanced in the NSL-KDD. The NSL-KDD also removed duplicate and redundant records.

Each sample records 41 features and two labels indicating whether the record is associated with an attack and the severity.

5.1.1. Classes of Attacks

Attacks are grouped into four classes: 1) denial of service (DOS), 2) probe, 3) user to root (U2R), and 4) remote to local (R2L). Attacks in each class are further classified into sub-classes (Saporito, 2019).

DoS attacks try to shut down traffic flow to and from targeted systems. The targeted system is flooded with an amount of traffic that is beyond what the system can handle. A successful DoS attack shuts down the system and prevents normal traffic from visiting a network. The probe tries to get information from a network. U2R tries to gain unauthorised privileged access to the system or network (e.g., as a root), while the R2L attack tries to gain local access to a remote machine. In this attack, an attacker who does not have local access to the system/network tries to gain a foothold in another connected network and traverse their way to the targeted system/network.

The subclasses are shown in the table below.

Classes:	DOS	Probe	U2R	R2L
Sub-classes	• Apache2 • Back • Land • Neptune • Mailbomb • Pod • Processtable • Smurf • Teardrop • Udpstorm • worm	• IPsweep • Mscan • Nmap • Portsweep • Saint • Satan	• Buffer_overflow • Loadmodule • Perl • Ps • Rootkit • SQLattack • Xterm	• FTP_write • Guess_passwd • Httptunnel • Imap • Multihop • Named • Phf • Sendmail • Snmpgetattack • Spy • Snmpguess • Warezclient • Warezmaster • Zlock • Xsnoop

5.1.2. Categories of Features

The recorded features are grouped into four categories: 1) intrinsic, 2) content, 3) time-based, and 4) host-based.

1) Intrinsic (features 1-9) These features are derived from the packet header without inspecting the payload.
2) Content (features 10-22) contains information about the original packets (i.e., the payload).
3) Time-based (features 23-31) These features contain information on the network traffic recorded over a 2-second window during the attack and contain information like how many connections it attempted to make to the same host. These features are mostly values like counts and rates rather than information about the content of the traffic.
4) Host-based (features 32-41) features are like Time-based features but contain information over several connections. They capture information on attacks that last more than two seconds.

5.2. Model Performance

For this set of experiments, the training set was split into two sets, the first set for the training of individual models, the second set for the training of stacking models. A set of six features were used ('protocol_type', 'service', 'flag', 'duration', 'src_bytes', 'dst_bytes').

The results were obtained by applying the model to the test dataset. The results are recorded in the table below.

	Accuracy	Precision	Recall	F1
(a) LogReg	0.786	0.639	0.977	0.773
(b) SVM	0.802	0.738	0.896	0.809
(c) a, b (stacked with RF)	0.812	0.735	0.919	0.817
(d) Bagging (RF)	0.819	0.712	0.959	0.817
(e) Boosting (Gradient Boosting)	0.833	0.734	0.964	0.834
(f) a, b, d, e (stacked with RF)	0.860	0.785	0.963	0.865

It can be observed that the SVM-based model outperformed the logistic regression-based one. Further performance gain was possible when they were stacked with a random forest model. Bagging and Boosting also provided significant gains. Stacking multiple models (as shown in f) can further improve model performance.

For the neural network-based experiments, the models were trained on the training set over several random seeds. The model performances over the different random seeds were averaged and reported below. This reduced the impact of the choice of seeds on the reported results. The results were obtained by applying the model to the test dataset. The following diagram shows the structure of the network used in the experiment.

	Accuracy	Precision	Recall	F1
(a) One hot encoding	0.789	0.662	0.956	0.781
(b) Label embedding	0.794	0.687	0.933	0.791
(c) Label embedding and smoothing	0.801	0.699	0.936	0.800

It can be observed that label embedding and smoothing helped improve model performance.

6. Conclusion

This paper discussed the different approaches to machine learning-based classification (e.g., logistic regression, support vector machines), their possible application, and the difficulties of using them on network intrusion detection. Though there are challenging problems to overcome before machine learning can be successfully applied to intrusion detection, there is an opportunity for network operators to add immense value to the network defense by collecting data and training intrusion detection models that are adapted to their networks. With a knowledge of the underlying algorithms and the provided source code, which applies some of the most recent machine learning algorithms, network operators can experiment with and eventually apply machine learning-based intrusion detection to their network.

Yih Han Tan, tanyihhan@gmail.com

References

Aurlien Gron. 2017. Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems (1st. ed.). O'Reilly Media, Inc.

C. Sinclair, L. Pierce, and S. Matzner, “An Application of Machine Learning to Network Intrusion Detection,” in Proc. Computer Security Applications Conference, 1999.

M. Mayhew, M. Atighetchi, A. Adler, R. Greenstadt. Use of machine learning in big data analytics for insider threat detection. In Proceedings of the MILCOM 2015-2

J. McHugh, “Testing Intrusion detection systems: A critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratories,” ACM Transactions on Information and System Security, vol. 3, no. 4, pp. 262–294, November 2000.

R. Sommer and V. Paxson, "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection," 2010 IEEE Symposium on Security and Privacy, 2010, pp. 305-316, doi: 10.1109/SP.2010.25.

K. Goeschel, “Reducing false positives in intrusion detection systems using data-mining techniques utilizing support vector machines, decision trees, and naive Bayes for off-line analysis,” Proceedings of the SoutheastCon 2016, Norfolk, VA, USA, 30 March–3 April 2016; pp. 1–6

K. Peng, V. C. Leung, Q, Huang, “Clustering approach based on mini batch k means for intrusion detection system over big data,” IEEE Access **2018**, *6*, 11897–11906.

R. Shah, Y. Qian, D. Kumar, M. Ali, M. Alvi, “Network intrusion detection through discriminative feature selection by using sparse logistic regression, “ Future Internet 2017, 9, 81.

R. Bapat, A. Mandya, X. Liu, B. Abraham, D. E. Brown, H. Kang, M. Veeraraghavan, “ Identifying malicious botnet traffic using logistic regression,” Proceedings of the

2018 Systems and Information Engineering Design Symposium (SIEDS), Charlottesville, VA, USA, 27 April 2018, pp. 266–271.

J. Long, Q. Liu, J. Cui, and W. Chen, "TR-IDS: Anomaly-Based Intrusion Detection through Text-Convolutional Neural Network and Random Forest," Secur. Commun. Netw. 2018.

H. Liu, B. Lang, M. Liu, H. Yan, "CNN and RNN based payload classification methods for attack detection," Knowledge-Based Syst. 2019, 163, 332–341.

M. Rigaki, S. Garcia, "Bringing a gan to a knife-fight: Adapting malware communication to avoid detection," Proceedings of the 2018 IEEE Security and Privacy Workshops (SPW), San Francisco, CA, USA, 24 May 2018, pp. 70–75.

KDD99 Dataset. 1999. Available online: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html (accessed on 19 May 2021).

NSL-KDD99 Dataset. 2009. Available online: https://www.unb.ca/cic/datasets/nsl.html (accessed on 19 May 2021).

G. Saporito, "A Deeper Dive into the NSL-KDD Data Set," https://towardsdatascience.com/a-deeper-dive-into-the-nsl-kdd-data-set-15c753364657, Sep 2019, (accessed on 21 May 2021).