Intrusion Detection Systems

Ioan Cristian Schuszter - **ioan.schuszter94@e-uvt.ro**

Metaheuristic Algorithms Report
West University of Timișoara, Romania
January, 2018

Abstract

Rapid adoption of the Internet into virtually all aspects of modern life (banking, health, education, research) has also raised many concerns in the framework of cybersecurity. As such, a set of mechanisms have been developed in order to protect hosts from a wide range of cyber threats. These include intrusion detection systems (IDS).

This paper studies various approaches of building an IDS, specifically the computational intelligence part of the systems. We start off by presenting an overview of the state of the art. The work then focuses on an implementation of an IDS, followed by a discussion of the results on a well-known dataset used for benchmarking IDS implementations, the KDD cup dataset from 1999.

Contents

1 Introduction **3**

1.1 Signature-based IDS 4

1.2 Anomaly-based IDS 4

1.2.1 Statistical-based methods 4

1.2.2 Knowledge-based methods 4

1.2.3 Machine learning-based methods 4

1.3 Machine Learning-based intrusion detection systems 5

1.3.1 Bayesian networks 5

1.3.2 Neural networks 5

1.3.3 Genetic algorithms 5

2 Methods used **5**

2.1 Feed-Forward Neural Networks 5

2.2 Self-Organizing Maps 6

2.2.1 Learning process 6

3 Dataset and performance evaluations used **8**

3.1 The KDD99 dataset 8

3.2 Performance metrics used 8

4 Implementation and Results **9**

4.1 Preliminary analysis and preprocessing 10

4.2 Self-Organizing Map classifier 10

4.3 Feed-Forward NN classifier 12

5 Conclusions and Open Problems **13**

1 Introduction

Intrusion detection systems are tools which are used in the field of computer security in the task of finding malicious actors or activity in a computer network or system, effectively strengthening these entitites. The field is very active and cyber security experts have been concerned about it for a few decades already. The basic structure and flow of usage for an IDS can be seen in **Fig**1.

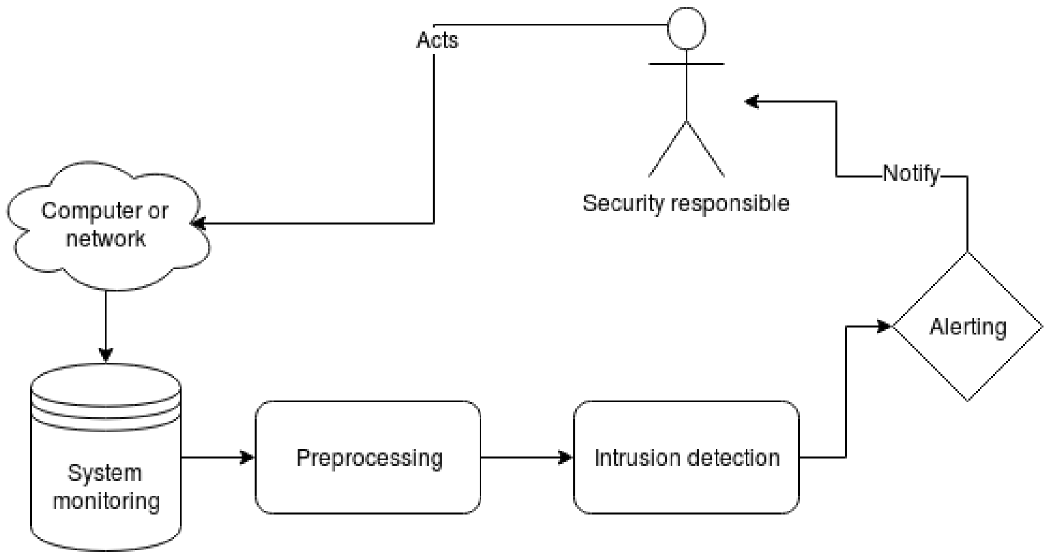


Figure 1: General overview of an Intrusion Detection System

IDS bibliography can be divided into a few major areas, according to the sources of the data and to the method used for detecting intrusions. One split can be done on the different types of components involved in the makeup of the system, as discovered by the Common Intrusion Detection Framework working group (CIDF):

- E blocks (event boxes) - these are the data sources fed into the system, being able to symbolize any type of log data or activity data
- D blocks (database boxes) - these components store te data from E blocks, optionally compressing and aggregating the data beforehand
- A blocks (analysis boxes) - they provide the intelligence behind the IDS and represent the focus of the research done in this paper.

Firstly, depending on the source of the information considered for analysis by the system, IDS can be split into:

- HIDS - host-based intrusion detection systems
- NIDS - network-based intrusion detection systems

The dataset in this paper is based on the second category of systems, the NIDS ones. As such, those systems will be the focus of this paper's research.

Based on the types of models that can be obtained, two large categories stand out as distinct:

1.1 Signature-based IDS

They focus on certain steps or actions that have been logged and are known to represent an attack scenario. These systems are the core of almost all of the commercial IDS systems that exist today, as they provide really good performance for known attacks and can generally be controlled and understood by domain experts. However, the major drawback of signature-based IDS is the fact that they can't learn to identify new types of attacks, and this is where the challenge of designing a good IDS appears, especially due to the fact that malware has taught itself to mutate a lot in the past decade.

1.2 Anomaly-based IDS

These IDS focus on learning from the data and establishing when a certain action or series of actions logged in the system deviate from normal type activity. The real attraction for these types of systems is the capability of constantly learning what is "normal" and what is not, taking into advantage the data that has been collected previously.

What happens is that the problem of IDS is reduced to a classification problem, one that separates normal activity from malicious. As such, this is the category that we will focus on in the paper.

In order for an anomaly-based network intrusion detection system to be built, there are generally a series of predetermined steps that need to be done. First, the data is parametrized, meaning that it is organized in such a way so that it can be trained on later. Afterwards, a model is trained based on the data and the resulting model is used for detection. A threshold is set for the value that the detector gives, and when that threshold is crossed an alarm is generated so that a human responsible for security should intervene.

A big advantage of these types of systems is the fact than any prior domain knowledge is not needed, as is the case with signature-base systems, the key being the fact that relationships emerge out of the data through data mining techniques.

The algorithms that form A-NIDS fall into three major categories that will be detailed in the following subsections.

1.2.1 Statistical-based methods

The oldest methods that were used come from this category. Z-scores and other univariate and multivariate methods of analyzing the data are computed from what was collected. When the scores of new input data deviate too far from the computed values, alerts are triggered and the action is considered malicious.

One drawback is the fact that attackers can actually figure out what the underlying model is and then simulate their traffic in such a way that the system is fooled, flagging the attack as normal.

1.2.2 Knowledge-based methods

These methods can rely on multiple things. Most commonly, they are represented by rule-based systems programmed by domain experts, which is also their largest problem. Other approaches use a domain language or finite state machines to represent the underlying model in the collected data.

1.2.3 Machine learning-based methods

These techniques represent the focus of the current paper and are covered by a comprehensive list of both supervised and unsupervised training models. The capacity of the techniques to create profiles of the users in the system is most appealing. However, the problem of having too many false positives appears, as every new type of behavior might be also tracked as being a deviation from the normal activity, hence an attack.

The second challenge posed by these techniques is the large volume of data needed to train a relatively robust model. This, coupled with the fact that network and activity logs from users are generally hard to use out of the box and need heavy preprocessing, makes these kinds of systems still be more research-oriented and not seen that much in the wild.

1.3 Machine Learning-based intrusion detection systems

This part will briefly cover the major available machine learning techniques used for intrusion detection.

1.3.1 Bayesian networks

These networks encode relationships between variables, being based on probabilities. It is generally used in combination with several statistical techniques, being much more accurate in that case. However, the results are not that better than using threshold based systems, while the computational costs are much greater

1.3.2 Neural networks

Similar to the way neurons in the human brain operate, they have been widely adopted in the field of anomaly detection, because of their capacity to adapt to new data. The main disadvantage of this approach is the fact that it acts as a black box, not offering any clear answer as to why the data was mapped like that.

Multi-layer feed forward neural networks and recurrent neural networks are frequently used for supervised learning, while there also are some methods based on unsupervised learning that have proven to be quite effective in classification of intrusions.

1.3.3 Genetic algorithms

These algorithms represent a class of search heuristics, the class of evolutionary algorithms. They use techniques inspired by biology such as mutation, inheritance and natural selection to produce models that are sound. The main problems of this method are high computational costs and possibility that the optimization might not converge to a global minimum.

2 Methods used

Two algorithms are used and compared in this paper. This section covers their usage and implementation.

2.1 Feed-Forward Neural Networks

Artificial neural networks (also known as connectionist systems) combine simple processing elements (called neurons) into an ensemble capable of delivering complex approximations of functions or classification, being inspired by the biological neural systems that constitute the human brain. Such systems learn (progressively improve their performance for a given task) to do tasks by looking at learning examples and being provided no explicit task programming needs.

Feed forward neural networks are the simplest form of modern-day neural networks, mapping a set of input data to a set of appropriate outputs. They are called feedforward because they contain no cycles, instead being a directed graph with each layer being fully connected to the next one, see **Fig.** 2. However, they work well for our implementation since the data is not that large and doesn't need a more complicated model.

The idea of feed-forward nets is that new data is classified by pushing data forward through the network and receiving a response on the last layer. The training process uses

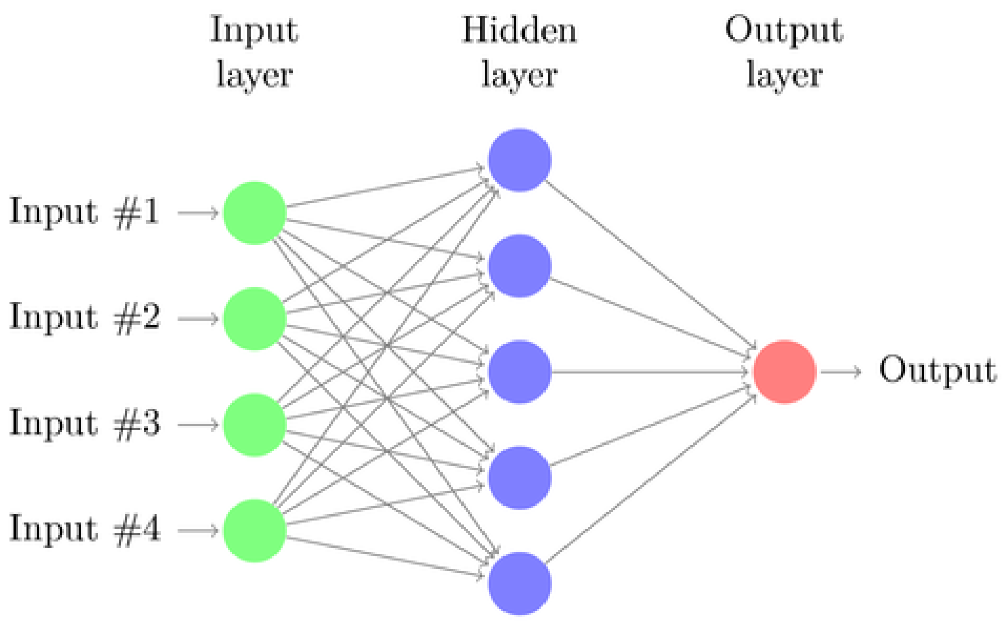


Figure 2: A feed-forward neural network

the **back-propagation** algorithm in order to adjust the weights of each neuron, yielding better predictions at each step.

2.2 Self-Organizing Maps

This type of algorithm belongs to the rare field of unsupervised neural network-based algorithms. According to [1], SOMs are the most popular neural networks to be trained for anomaly detection tasks. Due to this, they fit well into the task of building good IDS, as most data points are "normal", in real life scenarios.

They are feed-forward neural networks that are clustered in a low-dimensional grid (2D or 3D). The topology of the data is preserved, grouping the data points based on their similarity. It was first described by Finnish researcher Teuvo Kohonen, thus the popular name of Kohonen maps and is currently one of the most popular neural networks to be used for anomaly detection. [2]

An example of how a Kohonen map is structured can be seen in 3.

The neurons of a Kohonen map are organized in a grid-like structure, with full connectivity between neighboring neurons. The training process is based on competitive learning, meaning that the neuron closest to the input vector is declared as the "winning" neuron or BMU (best matching unit). Its weights are then adjusted towards the input vector, while also pulling its neighbors.

2.2.1 Learning process

The learning process of the SOM goes as follows:

1. One of the vectors from the input data is randomly selected and its similarity to the neurons' weights is measured by using Euclidean distance. Assuming the vector is x:

$$||x - W_u|| = min(||x - W_i||), \forall i = 0, n$$

2. After the BMU is found, the weight vectors are updated. The BMU itself as well as

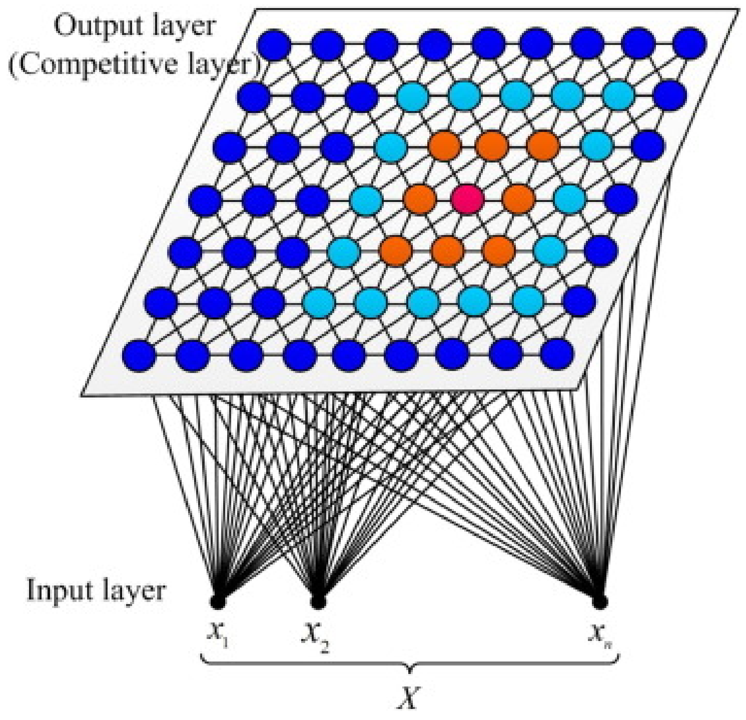


Figure 3: A visualization of a Kohonen map

the topological neighbors are moved closer to the input vector. The magnitude of attraction is given by the learning rate, which is one of the parameters of the algorithm.

As the learning progresses, the learning rate gradually decays.

The weight update process is:

$$W_v(s+1) = W_v(s) + \theta(u, v, s) * \alpha(s) * (x - W_v(s))$$

where the parameters are:

- W_v - weights of node v
- u - index of the BMU
- $\theta(u, v, s)$ - the neighborhood function
- $\alpha(s)$ - the learning rate at step s
- x - the target input vector

The steps 1 and 2 are repeated until s is reached, it being one of the parameters of the Kohonen algorithm.

The problem with using SOM for classification, after being trained, is the fact that multiple classes might be assigned to a single neuron. As such, one method for solving these ambiguities and using the network for classification is described by [3]. After training the data, each neuron is labeled according to the majority of the data type that fell into that category. If more then 50% of the entries were labeled as intrusions, any new mapping to that node would also be mapped as an intrusion.

3 Dataset and performance evaluations used

It is crucial in machine learning applications, and especially in applications that are critical such as IDS, to be able to have a good idea of the performance of the system. There have been some efforts to evaluate IDS data sources in the past [4], but valid and widely used data still remains scarce out in the wild.

Obtaining high-quality data is especially tricky in these systems since multiple sources need to be aggregated and some feature selection and extraction needs to be done.

We will discuss the famous dataset used for testing our implementation as well the performance metrics taken out of the results.

3.1 The KDD99 dataset

The KDD99 dataset is derived from the DARPA-Lincoln dataset collected by MIT's Lincoln laboratory, for DARPA Research Lab, including more than 300 instances of 38 different attacks grouped into 4 categories. The individual TCP packagets from the 1998 dataset were aggregated into TCP connections, resulting in the KDD99 dataset.

The KDD stands for the International Knowledge Discovery and Data Mining Tools Competition and it's the most popular dataset in intrusion detection, being used as a benchmark by many papers.

The dataset is comprised of the following 41 features:

- basic features: 9 features used to describe each TCP connection
- content features: 13 features related to domain knowledge
- time-based traffic features : 9 features that are used to describe the connections from the past seconds
- host-based traffic features: 10 features using a window of 100 connections

The training dataset contains almost 5 million instances, covering normal traffic and 24 attacks. Thus, there's a smaller dataset containing 10% of the whole data which is used more often.

The test data comprises around 300k instances and contains 12 attacks that the original dataset doesn't contain.

In the implementation and tests, the 10% dataset is used, in order to shorten the training time, especially in the case of the neural network. The main issue with training was the lack of adequate hardware, as only an i5 laptop processor was available.

3.2 Performance metrics used

IDS are mainly evaluated by their ability to make correct predictions and according to the nature of the responses, 4 cases exist:

- True positives - malicious activity flagged as malicious correctly
- True negatives - normal activity identified as normal
- False negatives - malicious activity flagged as normal (this is the most undesired property)
- False positives - normal activity identified as malicious

Some metrics based on the above definitions are, according to [1]:

- True Negative Rate:
$$TNR = \frac{TN}{TN + FP}$$
- True Positive Rate (also known as Detection Rate (DR) or Sensitivity or Recall):
$$DR = \frac{TP}{TP + FN}$$
- Accuracy:
$$Acc = \frac{TN + TP}{TN + TP + FP + FN}$$
- False Positive Rate (a.k.a. FAR - false alarm rate):
$$FAR = \frac{FN}{TP + FN}$$

The most popular metrics in the case of IDS are DR (detection rate) and false alarm rates. An IDS should have a high DR and a low FAR.

Some other popular combinations include precision and recall as well as roc curves, the receiver operator characteristic. As such, we will focus on these methods when evaluating the classification of the connections.

4 Implementation and Results

The core of the implementation is done in Python, using a number of popular libraries in order to achieve the goal of the paper. Two methods were implemented and tested, a comparison between them being available towards the end of this chapter.

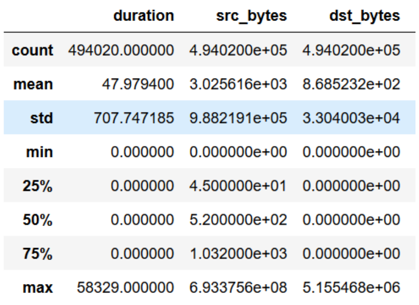
The following dependencies are needed for running the code:

- scikit-learn
- minisom
- matplotlib
- tensorflow
- pandas
- keras
- graphviz

The structure of the code is as follows:

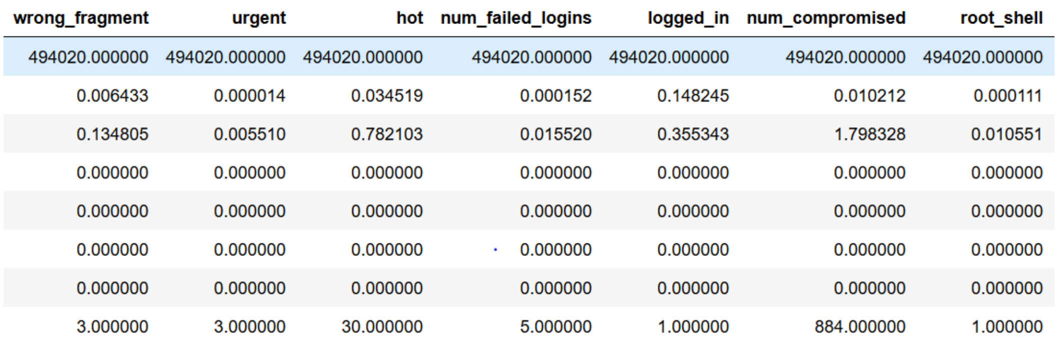
- 2 Jupyter notebooks containing the implementations and plots presented in the paper : "som_implementation.ipynb" and "nn_implementation.ipynb"
- A library needed for common metrics and utils: "ids_metrics.py" and "ids_utils.py"

There is also a data folder that provides the 10% of the KDD dataset as well as 10% of the final testing data.



	duration	src_bytes	dst_bytes
count	494020.000000	4.940200e+05	4.940200e+05
mean	47.979400	3.025616e+03	8.685232e+02
std	707.747185	9.882191e+05	3.304003e+04
min	0.000000	0.000000e+00	0.000000e+00
25%	0.000000	4.500000e+01	0.000000e+00
50%	0.000000	5.200000e+02	0.000000e+00
75%	0.000000	1.032000e+03	0.000000e+00
max	58329.000000	6.933756e+08	5.155468e+06

Figure 4: Variables that are distributed over a large space



wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromised	root_shell
494020.000000	494020.000000	494020.000000	494020.000000	494020.000000	494020.000000	494020.000000
0.006433	0.000014	0.034519	0.000152	0.148245	0.010212	0.000111
0.134805	0.005510	0.782103	0.015520	0.355343	1.798328	0.010551
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
3.000000	3.000000	30.000000	5.000000	1.000000	884.000000	1.000000

Figure 5: Variables that are distributed over a small region

4.1 Preliminary analysis and preprocessing

Loading the 10% version of the KDD dataset, the dataset is found to contain 42 columns, the last one being the label for that specific connection.

Upon preliminary description of the dataset, a very big difference between the variables can be seen by comparing 4, 5. As such, normalization of the data seemed necessary before doing any processing or training in a classifier, especially in the case of the Neural Network implementation.

Afterwards, the difference between normal activity and the majority of attacks, with respect to the number of connections, is seen in 6. It shows that intrusion detection is indeed a particular case of anomaly detection.

There are also a few categorical variables in the dataset, describing details about the connection, such as the protocol. They were chosen to be mapped from their categorical implementation to a list of dummy variables, one for each level of the factor. For example, in the case of **protocol_type**, the variable was mapped to three columns, as you can see there are only 3 values in the dataset, in 7.

4.2 Self-Organizing Map classifier

The implementation of Self-Organizing Maps in the case of our paper, uses a small but popular and efficient Python implementation called MiniSom [5]. We chose a grid of 20x20 neurons for the classifier, with a sigma of 0.3 (spread of the neighborhood function) and a learning rate of 0.5. The larger the learning rate, the more the data would be spread out. The algorithm was chosen to run for 10000 iterations, until stopping, as there are no early-stopping methods available.

Some post-processing is also needed in order to get the Self-organizing map to be used for classification. Once the network is trained, a matrix of maps from string to integer

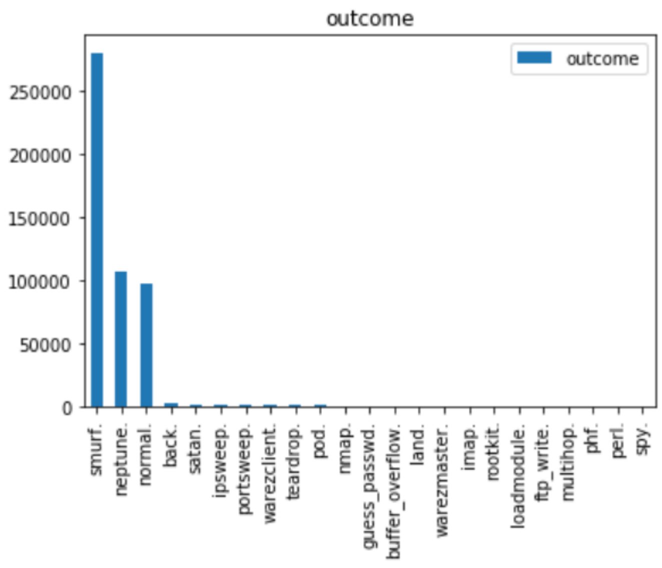


Figure 6: Distribution of outcome (the class label)

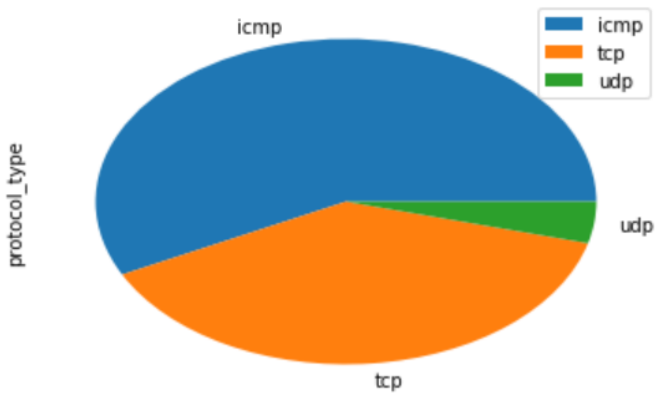


Figure 7: Pie-chart for protocol_type

needs to be created.

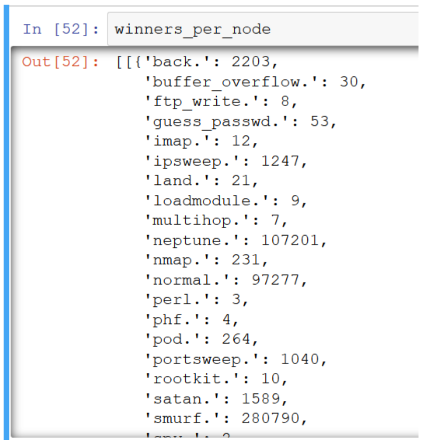
These maps record how many connections of each class are mapped to each neuron. Then, the method described in [3] for deciding the threshold of a neuron is used: if the sum of all attacks mapped to a neuron is larger than 50% of the entire neuron's mappings, the node is classified as "malicious", otherwise "normal".

Thus, the multi-class classification problem is transformed into a binary classification problem where all of the attacks are added to the category of "malicious".

Then, the system is scored using the metrics described in **3.2**.

After performing the preprocessing, one unsuccessful attempt was made, using the completely normalized dataset. However, the classifier was not sensitive enough to separate the data into different neurons, ending up mapping everything to the first neuron and leaving the empty neurons with no mappings, as seen in 8. This most likely happened due to the fact that SOMs map from high-dimensional space to 2D space and some distance might be required for it to work correctly.

After concluding that normalizing the data is not the right way to go, the dataset was taken "as is", so that it could be validated against the test data. Using a train-test split of



```
In [52]: winners_per_node

Out[52]: [[{'back.': 2203,
            'buffer_overflow.': 30,
            'ftp_write.': 8,
            'guess_passwd.': 53,
            'imap.': 12,
            'ipsweep.': 1247,
            'land.': 21,
            'loadmodule.': 9,
            'multihop.': 7,
            'neptune.': 107201,
            'nmap.': 231,
            'normal.': 97277,
            'perl.': 3,
            'phf.': 4,
            'pod.': 264,
            'portsweep.': 1040,
            'rootkit.': 10,
            'satan.': 1589,
            'smurf.': 280790,
```

Figure 8: Mapping of all instances to the first neuron

80/20 on the train and using the entire 300k+ data points from the test data as validation, the results were:

Dataset	Instances	Corr. classified	FAR	DR	Accuracy
Test	74103	73664	0.56%	99.43%	99.40%
Validation	311028	284559	10.40%	89.59%	91.48%

What is interesting is the fact that the validation dataset contains information about attacks that are not present in the original dataset. In state-of-the-art commercial IDS systems, a FAR of about 3%-4% is the norm, and obtaining such a good accuracy for just 10% of the dataset is deemed acceptable.

4.3 Feed-Forward NN classifier

The Feed-Forward NN is a rather standard approach in multiclass classification. We employed both normalization of all the numeric data points and the transformation of the categorical datapoints into new variables.

The total number of resulting variables resulting is 119 for the training dataset and 118 for the test dataset (a category seems to be missing in the test set). Due to this, validation is performed on a piece of the training dataset, leaving out the test set completely.

The architecture of the network is as follows:

- 4 densely connected hidden layers, with the following makeup: 10, 15, 10, 1
- the activation function for the hidden layers is the "relu" function
- the output layer has 3 neurons and a softmax activation function. The neuron with the greatest response is declared the winner
- The loss function is the categorical crossentropy, as it works well for classification tasks
- The optimization algortihm is the ADAM [6] optimizer, as it's been proven to be quite fast
- The train/test split was 75% / 25%

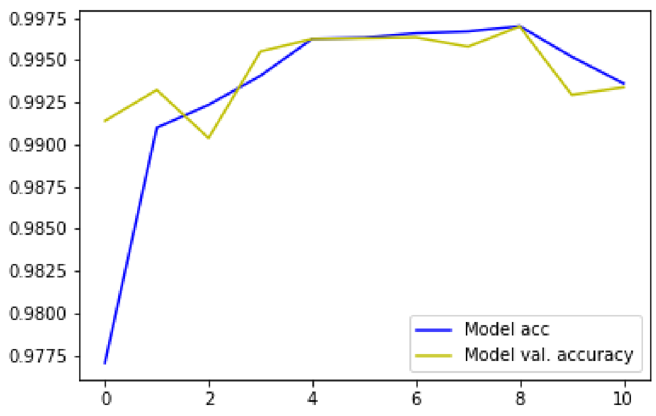


Figure 9: Accuracy graph

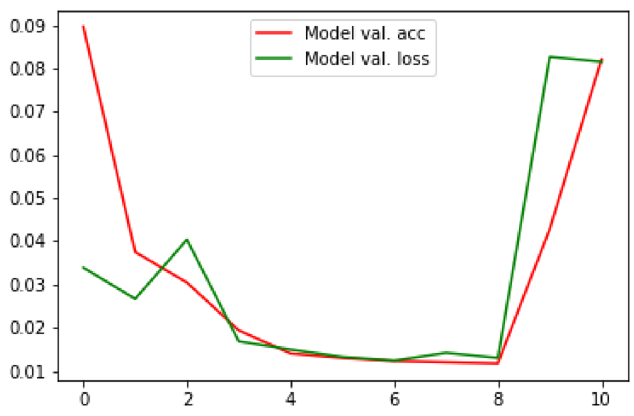


Figure 10: Model loss graph

Also, an early stopping criterion is used in order to prevent overfitting of the dataset. The classifier stopped after 11 epochs, as the data seems to be easy to classify in the case of neural networks. Plots 9, 10 show the accuracy and loss, respectively, of the model for each epoch.

After obtaining the validation metrics, the following table results:

Dataset	Instances	Corr. classified	FAR	DR	Accuracy
Test	123505	122688	0.48%	99.51%	99.33%

5 Conclusions and Open Problems

In conclusion, there are still some unsolved issues due to the difficult nature of obtaining a valid IDS dataset that can be used to train classifiers. In most cases, results published don't provide the dataset, as it may have sensitive information inside.

Regarding the performance of the classifiers, both of them have been shown to perform well on the KDD99 dataset, for this specific tasks. However, there is no knowledge as to how the systems might perform in real life. The SOM implementation gets close to being exposed to real world data, through the validation dataset that contains previously unseen attacks.

Regardless of the implementation chosen, both systems offer very good accuracy and FAR on the testing data. However, the SOM extra post-processing times are time consuming compared to that of the simple neural network.

References

[1] Shelly Xiaonan Wu and Wolfgang Banzhaf. Title: The use of computational intelligence in intrusion detection systems: A review. *Applied Soft Computing*, doi:10.1016/j.asoc.2009.06.019, 2009.

[2] T. Kohonen. The self-organizing map. *Neurocomputing*, 21(1):1-6, 1998.

[3] John Zhong Lei and Ali Ghorbani. Network intrusion detection using an improved competitive learning neural network. *Faculty of Computer Science, University of New Brunswick.*

[4] A. N Zincir-Heywood H. G Kayacik and M. I Heywood. Selecting features for intrusion detection: A feature relevance analysis on kdd 99 intrusion detection datasets. *Proceedings of the third annual conference on privacy, security and trust*, Citeseer, 2005.

[5] G Vettigli. Minisom: Minimalistic and numpy based implementation of the self organizing maps. `https://github.com/JustGlowing/minisom`.

[6] Johnson Apacible Trishul Chilimbi, Yutaka Suzue and Karthik Kalyanaraman. Project adam: Building an efficient and scalable deep learning training system. *11th USENIX Symposium on Operating Systems Design and Implementation*, 2014.