Application of Machine Learning Algorithms in Cyber-Security

Shawn Saliyev
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA 56267
saliy002@morris.umn.edu

ABSTRACT

Malicious software is one of the main and continuous sources of threat in the world of cyber-security. More innovative ways are needed for detecting them. This paper investigates two different machine learning approaches for detecting malicious attacks. The first approach is a combination of two types of neural networks called autoencoder and deep neural network. The second approach uses a combination of a genetic algorithm and an isolation forest. We will be going over these two approaches and sharing results that describe how effectively these methods detect malicious softwares.

Keywords

Machine Learning, Isolation Forest, Deep Learning, Deep Neural Network, Cyber Security, Cyber Threats

1. INTRODUCTION

In the modern day it, is hard to imagine our lives without a computer or phone that is connected to the internet. We use these devices on an everyday basis in order to communicate, learn, and work. Because of that lifestyle we end up storing, sending, or receiving a lot of personal or valuable data. And, unfortunately, there are people who want access that data and either steal it to use it against the owner. For that they develop malicious software (malware) that if executed on your system can damage or send your information to the developer of that program. The issue is that it is not trivial task to detect such malicious files. By downloading a file from a website, you might end up receiving malware.

One of the old and traditional approaches that has been used for detecting malware is a *Signature Based Detection System*. This technique involves having an updated database with unique byte or instruction patterns that are referred to as signatures. Each signature is used as an identifier for specific malicious software. When checking an unknown downloaded file, the system simply matches it against the database and if a match is found then the file is classified as malicious. The problem is that this approach is really inefficient against new malware referred to as Zero-Day attacks. However there are other new methods that address this issue.

Relatively new ways of detecting malicious software involve using different machine learning algorithms. Machine learning algorithms tend to be really efficient for this purpose. One of the big cyber-security companies called "Kaspersky Lab" applies some machine learning algorithms such as decision tree ensembles, behavioral models, and clustering [1]. This paper will focus on two other techniques. The first is based on using an autoencoder for extracting the important details about the data and deep neural networks for classifying the files into either safe or malicious. second technique uses a genetic algorithm for determining important features of the data and an isolation forest for detecting malware.

This paper will define some machine learning terms such as deep neural network, autoencoder, isolation forest and genetic algorithm in the next section. Background information about malicious software will be also provided in the next section. In Section 3, we will introduce the deep learning approach and look at its result. Section 4 will be dedicated to the Isolation Forest approach. We will come up with the conclusions in Section 5.

2. BACKGROUND

There are some key concepts that need to be elaborated upon before the investigation of these two methods. First, we define the concept of machine learning itself and then go over main terms. Next we describe definitions and notations necessary to understanding the idea of the malicious software.

2.1 Machine Learning

Machine learning (ML) is an approach that is successfully used in many areas such as cyber security, medicine, and engineering. Generally the goal is to generate a *model*, which can be some math representation of some process. It is really efficient and practical because it allows us to teach the computer to do a specific task based on the data. By learning we mean identifying patterns and features in the data.

2.1.1 Data

It is really important to understand what we mean by *data* in the context of machine learning. Data, or as it is more commonly referred to *data set* is an essential part of machine learning, since it is used to teach the computer to do a specific task. The size of the data set is one of the parameters that affects the total performance of a machine learning algorithm. Typically there are two types of data sets: labeled and unlabeled. Labeled data set is where the label for each input is already provided. Unlabeled data set

UMM CSci Senior Seminar Conference, December 2018 Morris, MN.

does not have labels, only input data. Data that is labeled is actually expensive and requires more effort to get than unlabeled data. In the context of malware detection you can think of a labeled data as a set of executable files which have labels that tell if the file is benign or malicious. The labeled data set is used for *Supervised Learning* technique, which will be described later.

There is also another key concept about data besides the size of the data set. If you think of a set of executable files, there are other parameters that describe each file individually. These parameters are referred as inputs. The correlation between inputs or inputs themselves sometimes can be referred to as *features*. For a simple example imagine a java executable file and two features that describe it: file size and execution time. For an example let's say file with the size of 1000 bytes has execution time of 2 seconds. Then this file can be described as a vector [2 1000], where the first entry corresponds to the execution time and the second entry is the file size. In the real world data is described by many features, so the size of such a vector could be much larger.

2.1.2 Techniques

A supervised learning technique is when your machine uses a labeled data set for learning. Since the data is labeled, the machine is looking for combinations of features of each data point and tries to associate them with the corresponding label.

The second technique is called unsupervised learning which uses unlabeled data. Since there are no labels provided, the machine can not really classify the data points. Instead it tries to cluster them according to their features.

The learning process is usually called *training*. During the training process the algorithm learns the pattern in the data. In case of labeled data, the algorithm is looking for the relation between input and output. If the data is not labeled, then algorithm is trying to group inputs together, such that inputs in the same group are similar to each other. The data set is divided into two uneven parts. For example 2/3 of the data set is used for training and the rest (1/3) is for testing. Testing is really important in machine learning. After training the algorithm on the training data set, we need to check how this algorithm will perform on the data that it has never seen before. [7]

2.2 Malware

Malware is short for malicious software [8]: a program that has been made for dealing damage to any kind of systems. Malicious software was intentionally made for dealing damages such as spying, corrupting data, deleting important files, and infecting other files or systems if reachable.

There are some common categories of malware: Trojan, Virus, Spyware etc. Trojan is one of the most popular malicious programs. It usually gets into the user's system by disguising itself as a safe program. Spyware is made for spying on a user. It is a really dangerous kind of malware since it can save all of your passwords and other important credentials. Viruses are good at spreading and corrupting files in your system, and there is also a risk that they can infect other systems that are on the same network.

3. DEEP LEARNING EXPERIMENT

In this section we will describe an approach that uses an autoencoder for extracting important features from the data set and deep neural network for classification. We will also look at the application and the result of the current method.

3.1 Neural Network

A neural network is the machine learning approach that might remind us of a human brain. It is a system that is efficient at learning structures and patterns of the data. Like a human brain, neural network finds and learns patterns in the data, and then uses this information to do a specific task. Neural networks are really convenient for classification tasks. So in the case of detecting malicious files, this approach becomes really useful.

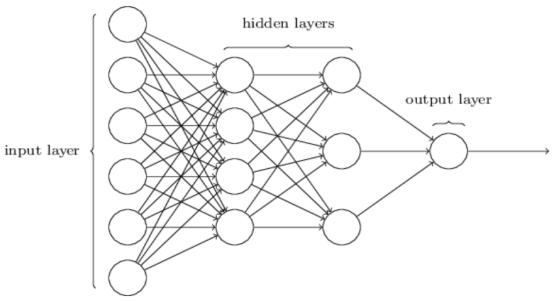

Figure 1: Example of the Neural Network[1]

In figure 1 you can see an example of neural network that has an input layer, two hidden layers, and an output layer. Technically this is a deep neural network since it has more than one hidden layer. Each layer has a certain number of nodes. This number is arbitrary and selected on a case by case basis.

3.1.1 Activation

Each node in the hidden layer has a nonlinear function called *activation functions*. Which defines the output of that node by taking in the input from the previous layer. An example of activation function might be exponential linear unit (ELU):

$$F(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^x - 1), & \text{if } x < 0 \end{cases}$$

Where α is some constant and e^x is the exponential function.

All nodes are connected with other nodes from neighbour layers. These connections are called *edges*. Each edge has the assigned *weight* to it. The weight is used for amplifying or silencing the output of the activation function.

In the figure 2, you can see a closer image of Neural Network. All inputs in the input layer are labeled as X_i where i is the index of the node in the input layer. Weights are labeled as W_i. The activation function takes in a weighted sum of inputs plus *bias*. Bias is needed for shifting the output of the activation function. The input of the activation function can also be written as:

$$\sum_{i=1}^{n}(X_i * W_i) + b$$

Where b is the *bias*. The output of the activation function is then used as an input for nodes in the next layer.

[1]http://neuralnetworksanddeeplearning.com/chap1.html

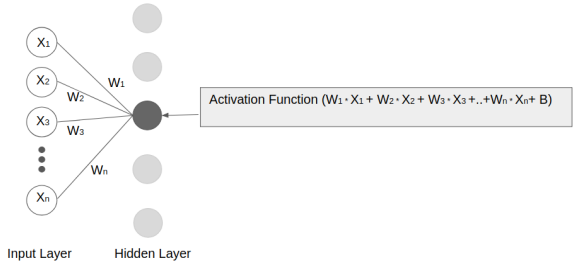

Figure 2: Closer look at the Neural Network

3.1.2 *Loss Function*

Initially each weight and bias is set to some random number. Therefore at first the performance of your neural network is really poor. In order to increase the performance, the weights and biases need to be adjusted in a right way. In order to understand how to adjust them a *loss function* is used. You can think of a loss function as the gap between the label and the output that has been produced by your network. So the main goal here is to find minima of the loss function or in the other words, to reduce the gap between the expected output and the output given by your network. If you are making the neural network that has to classify the data between two classes as in our case, malicious or benign, then it would make sense to use the loss function that fits the binary output. An example of such loss function is *Binary Cross-Entropy Loss Function*. This loss function can be computed as

$$L = \frac{1}{n}\sum_{i=1}^{n} y_i \log\left(\frac{1}{p(x_i)}\right) + (1-y_i)\log\left(\frac{1}{(1-p(x_i))}\right) \quad (1)$$

The y_i represents the actual output, for example if the file x_i is malicious then y_i is 1 and 0 if otherwise. The $p(x_i)$ represents the output that has been predicted by the network, it is presented as a function of $p()$ because the output layer of the neural network does not really give 0 or 1 as the output. The output is given as certainty or probability. For an example if the output of the network is 0.8, that would mean the 80% of chance that current file is malicious. The $y_i \log(\frac{1}{p(x_i)})$ part represents the loss or the gap between expected output y_i and predicted by the network model $p(x_i)$. By looking at this formula you can see that closer y_i is to $p(x_i)$ then the total of $y_i \log(\frac{1}{p(x_i)})$ becomes less. In the case when y_i is 0 (benign program) the $y_i \log(\frac{1}{p(x_i)})$ becomes 0 as well, that is why there is a second term in the sum $(1-y_i)\log(\frac{1}{1-p(x_i)})$. So the equation 1 represents the combined gap between all y_i and $p(x_i)$.

3.1.3 *Optimization*

Optimization is an essential part of training a neural network. The goal is to minimize the loss function. *Gradient Descent* is the optimization algorithm that is used for finding such local minima of the loss function. The algorithm is based on calculating the gradient of the function. Where gradient is the vector-valued function which shows the direction of the greatest increase of a multivariate function. Hereby the algorithm moves in the direction of steepest descent by taking the negative of the gradient. Figure 3 shows how gradient descent algorithm moves downhill, until it reaches the local minima. You can think of a gradient as a general form of derivative of an function, where derivative is a scalar-valued function because it is used when there is only one variable.

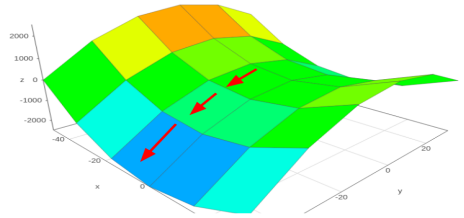

Figure 3: Multi-variable function [2]

For the simplicity, let the loss function be L and weight be w. So the goal is to find w^* such that $L(w^*)$ is less than $L(w)$ for all other w. In order to understand the direction to the optimal w^*, the gradient function is computed.[2]

The process of finding such minima is iterative and can be shown as

$$w_{i+1} = w_i - a_i G_L(w_i)$$

Where i is the number of the iteration, w_i is the current weight, and $G_L(w_i)$ is the gradient of the loss function with respect to that weight. The learning rate a_i is a positive scalar that determines the step size of the gradient descent algorithm. The a_i depending on the optimization algorithm can be same through all iterations, or it can get smaller with each iteration.

3.1.4 *Regularization*

There is always a risk that the model that we are training with the training data set, can become too specific for that data. This condition is called *overfitting*. The network learns the relationships in training data set so well, that when we test it with the testing data, it performs poorly. One of the reasons is that only portion of weights have been modified during the training process. That way the other portion of weights is left unchanged.

Regularization is used to prevent overfitting of the neural network. There are a lot of different regularization techniques that are being used. An example of an regularization could be *dropout*. Dropout is a technique that turns off certain percentage of random nodes during the training process. That way it makes all weights to be changed during training and decrease the chances of overfitting.

3.2 Data Gathering

The dataset for this experiment was collected from two different sources. Malicious binary files were extracted from Malicia project, and benign programs were collected from different Windows systems. The total size of the dataset was about 14,000 files where almost 80% were malicious files and the rest were benign programs. Because of that big imbalance, the Adaptive Synthetic Oversampling technique was applied. This technique generates more synthetic samples of benign files and keep the distribution uniform. After that

[2]https://academo.org/demos/3d-surface-plotter/

each of the files was used to generate their assembly code files.[6]

In order to prepare data for the neural network and have a list of features as representation of each assembly file, the *opcode frequencies* were used. [5] The *opcode* is a single instruction that machine can understand and execute. An example of opcode can be "ADD CX,BX" where *ADD* is the instruction, *CX* and *BX* are operands. First the set of unique opcodes was extracted from the assembly files. In total there were about 1600 unique opcodes. Then the set was numbered so that it could be used as table where their frequencies can be recorded. Then each file was checked, and the table corresponding to that file was filled with frequencies of each opcode. These frequencies were scaled from 0 to 1. These were used as inputs.

3.3 Feature Extraction and Dimensionality Reduction

Having about 1600 features for each data point would mean that network will have to work with 1600 dimensional data. The complexity of data significantly increases with the number of dimensions. However we can reduce the number of dimensions by finding features that are correlated or overlapped. Therefore some can be removed since they will not provide extra information about the data. For that autoencoder was used. Autoencoder is the type of neural network that can be used for extracting important features and reducing the dimensions of the data. For this experiment single-layer and 3-layer Autoencoders were used. [6]

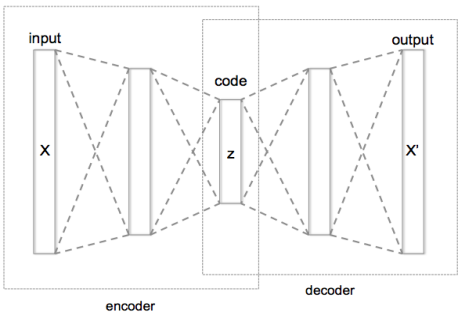

Figure 4: Structure of the Autoencoder

The structure of the single-layer Autoencoder (1L-AE) consists of input layer, bottleneck layer, and output layer. The 3-layer Autoencoder (3L-AE) has two extra layers before and after the bottleneck layer. The bottleneck layer of this autoencoder consisted of 32 nodes. Hereby there are two main components: *encoder* and *decoder*. Where encoder consists of layers before the bottleneck layer and decoder consists of layers after the bottleneck layer.The encoder compresses the input layer into the bottleneck layer by combining the common features into one element. Then the decoder finds the relations between the features that describe the expected output.

3.4 Deep Neural Network models

For the classification part, three different deep neural networks were used: 2-hidden layer (2L-DNN), 4-hidden layer (4L-DNN) and 7-hidden layer (7L-DNN). The classification model takes in the features from the bottleneck layer that was made by autoencoder as an input.

All three of these networks use ELU as the activation function except the output layer. The output layer consists only of one node and uses Sigmoid as the activation function. The sigmoid function is often used for the output layer of networks that do classification between two categories.

For the regularization the 0.1 dropout technique was used, where 0.1 means that random 10% nodes will be turned off during the training process in every iteration. The binary cross entropy loss function was used as loss function. Adam optimizer was used for the optimization of the neural network. Adam is another type of optimization algorithm that has adaptive learning rate. Which means that the learning rate is being change during training process. This optimizer is really popular and efficient, that is why it is commonly used for neural networks. [4]

3.5 Results

The overall results of this experiment are quite impressive. The chart below demonstrates the performance of different combinations of Autoencoders with Deep Neural Networks.

The combination of 3-layer autoencoder and 4-layer deep neural network turned out to be the most accurate (99.21%). The combination of 3-layer autoencoder and 7-layer deep neural network showed the lowest accuracy (93.60%). One of the reasons why the combination of 3-layer autoencoder and 7-layer deep neural network showed worst performance, is that the complexity of the model that is being generated by the network is high, because the big number of hidden layers. This issue is called *overfitting*. Table 1 shows the results with different combinations of autoencoder and deep neural network.

- **TP** - true positive, malicious file identified as malicious.
- **FP** - false positive, benign file identified as malicious.
- **TN** - true negative, benign file identified as benign.
- **FN** - false negative, malicious file identified as benign.

AE	DNN	TP	FP	TN	FN	Acc.
1L-AE	2L-DNN	3481	23	3612	200	96.95
1L-AE	4L-DNN	3451	44	3591	230	96.25
1L-AE	7L-DNN	3618	11	3624	63	98.99
3L-AE	2L-DNN	3630	157	3478	51	97.16
3L-AE	4L-DNN	3630	7	3628	51	99.21
3L-AE	7L-DNN	3238	25	3610	443	93.60

Table 1: Results with different combinations of AE and DNN [6]

4. ISOLATION FOREST

This section addresses the second experiment where Genetic and Isolation Forest algorithms were applied for detecting malicious files.

4.1 Isolation Forest

An isolation forest (IForest) is another type of machine learning algorithm which is convenient for identifying anomalies in data. Where by anomaly we mean the data point that

is not similar to the majority of the data. As an example, imagine a set of 10 benign executable files and couple of malicious files. Malicious files will have different features in this set, thus can be identified as anomalies. There are two main parameters for this algorithm: the number of isolation trees and the sub-sample size.

Sub-sample size determines the number of data points that will be randomly chosen for generating each isolation tree.

An Isolation tree is what is used to create the isolation forest. It is structured as binary search tree where values less than some specified value end up on the left side of the tree and other values build the right side of the tree. In order to build such a tree, the algorithm randomly chooses the dimension and a random value x of that dimension from a given data set. Then all data points that have smaller value than x in that dimension build the left side of the tree. Data points that have greater or equal value than x in that dimension build the right side of the tree. That process will be repeated until every data point is separated. Figure 5 shows the process of generating the isolation tree.

Figure 5 shows that the anomaly point is closer to the root of the tree compared to the majority of other data points. That happens because it takes less splits to separate such outlier. Since the building of the tree involves randomly choosing values, it would not make sense to build just one such a tree. That is why the IForest algorithm builds a set of trees.

After all trees are generated, the average path length from the root to each node is calculated. The node which has short average path to the root among all trees is identified as an anomaly.

In this experiment the number of isolation trees and sub-sample size were selected by trial and error. Initially the number of isolation trees was 100 and sub-sample size per tree was 256. But after trying different parameters,the authors ended up using 200 isolation trees and sub-sample size was 8192. [3]

4.2 Genetic Programming

Genetic programming is one of a kind of evolutionary algorithms. The main goal is to evolve the computer program so that it solves the given problem. This algorithm is based on Darwin's theorem of evolution which says that the fittest individual will survive in the given environment. The process of evolving the programs is called an evolutionary run. At the beginning of each run there is a set of randomly generated individuals that can potentially solve the given problem. All individuals are tested against a given test and receive the score that describes how fit are they for solving a problem. Better individuals are selected to be parents of the next generation. The next generating can be produced by two main ways: crossover and mutation. Crossover is similar to sexual reproduction where child program inherits its parts of the program from both parents. The mutation happens when parts of the parent's program are randomly changed and used for making the child program. The process of producing new generations will continue until the solution is found or a specific score is reached.

4.3 Data Gathering

For this experiment malicious files were created based on the examples from "ContagioDump" and "Malware Domain List" websites. Benign files were made using Sqrrl's attack lab enterprise. This enterprise is mid-sized and hosted by the company called Simspace. It is a simulation system which has network hardware and endpoint machines for simulating normal business activity. Then this network activity has been recorded and used as benign files. Both data sets consisted of Packet Capture (PCAP) data. This data has been decoded into Bro HTTP log files with the use of "Bro" software. At the end there were 37,978 malicious and 271,129 benign HTTP log files.

HTTP log files have multiple fields that describe the HTTP request. An example of such fields could be field called *host*, which tells the domain's name. The problem is that most of these fields such as *host* have string values, therefore they need to be represented as numbers so that IForest algorithm can use them as dimensions. Couple of techniques such as *bag of words* and *N-grams* were used to address this problem. N-gram features were produced by extracting n-consecutive character sequences and and group them. Each of these groups will represent its own dimension, but with only two possible values 0 and 1. Then if the current log file contains such character sequence it gets value of 1 in this dimension. The bag of words operates in the same way except instead of character sequences it just groups single words. [3]

4.4 Feature Extraction

Bro log files from initial data set have 22 fields. That means there are already 22 dimensions, but since most of the field have string values, the fields have multiple dimensions as well. Because of the time constraints, it was decided to reduce the number of fields by choosing more important ones. The check of each combination of fields just to find more important ones would require a lot of time. Thus genetic programming algorithm was applied in this situation. Because the genetic algorithm focuses on finding the fittest individual for a specific problem, it became useful in finding a subset of fields that would provide best performance for IForest algorithm.

The population size for each generation was set to be 25 individuals. Each individual has 20 booleans that represent the log file. Each of these booleans would correspond to a field from the log file, then if this individual had this field the boolean value will be set to true otherwise it will be false. That is each individual would represent the subset of fields which will be used by IForest algorithm. Once IForest is done at identifying the anomaly points in the same data set considering only current individual's subset of fields, the score of overall performance is attached to that individual. Then that score is used by genetic algorithm to identify most successful individuals so that they can be selected as parents for next generation.

The score in this experiment was evaluated in terms of area under receiving operating characteristic curve(AUROC). The receiving operating characteristic curve is the plot of true positive rate (TPR) against false positive rate (FPR) at different threshold values. The TPR is calculated by dividing the positive correctly classified by total positives (P) and FPR was calculated by dividing negative incorrectly classified by total negatives (N).

$$TPR = \frac{TP}{P}$$
$$FPR = \frac{FP}{N}$$

The Figure 6 demonstrates an example of the ROC curve.

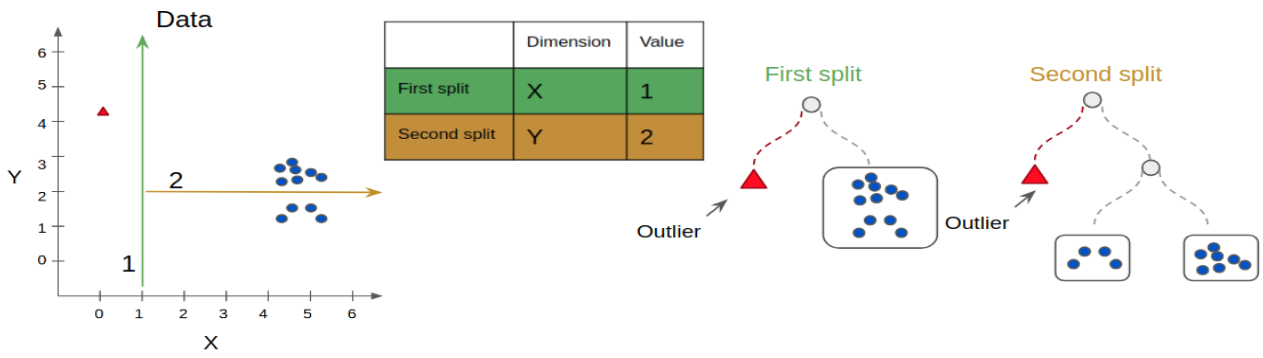

Figure 5: The process of generating the isolation tree

The area under that curve in the specific range is called AUC. That value determines the success of the IForest algorithm, the goal in this experiment was to increase the area under the ROC in between 0.1% and 1% false positive range. The reason for that limit is that it would be harder to tolerate the false positive range higher than 1%. The greater the area under the red curve, means better performance of the algorithm.

The genetic programming algorithm was running for 20 generations, then the top 5 individuals who had the greatest AUROC score were selected. The fields which were in all or most of these 5 individuals were selected as features for the IForest algorithm.

4.5 Results

After identifying the most important features and testing different number of isolation trees and different subsample sizes, authors got significantly good results. The 6 demonstrates the ROC, the area under red curve is equal to 0.00828.

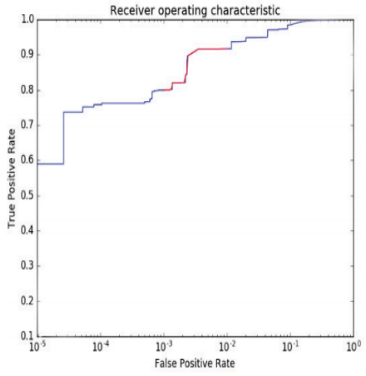

Figure 6: The ROC curve that represents the performance of the Isolation Forest algorithm

5. CONCLUSION

This paper investigated to different approaches of identifying malicious softwares. The first approach is the combination of two different neural networks. Another approach is the combination of isolation forest and genetic algorithm. Results demonstrated that with adjusted parameters these machine learning algorithms are very effective against malwares. This investigation shows that there is still plenty of room for exploration, different combinations provide different results, some of them are very promising.

6. ACKNOWLEDGEMENT

Thank you to KK Lamberty, Elena Machkasova, and Sydney Richards for advising and feedback

7. REFERENCES

[1] Kaspersky lab, 2017. https://media.kaspersky.com/en/enterprise-security/Kaspersky-Lab-Whitepaper-Machine-Learning.pdf.

[2] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[3] D. Karev, C. McCubbin, and R. Vaulin. Cyber threat hunting through the use of an isolation forest. In *Proceedings of the 18th International Conference on Computer Systems and Technologies*, CompSysTech'17, pages 163–170, New York, NY, USA, 2017. ACM.

[4] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[5] I. Santos, F. Brezo, B. Sanz, C. Laorden, and P. G. Bringas. Using opcode sequences in single-class learning to detect unknown malware. *IET Information Security*, 5(4):220–227, December 2011.

[6] M. Sewak, S. K. Sahay, and H. Rathore. An investigation of a deep learning based malware detection system. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, ARES 2018, pages 26:1–26:5, New York, NY, USA, 2018. ACM.

[7] Wikipedia contributors. Machine learning — Wikipedia, the free encyclopedia, 2018. [Online; accessed 30-October-2018].

[8] Wikipedia contributors. Malware — Wikipedia, the free encyclopedia, 2018. [Online; accessed 11-October-2018].