One-Class Classification of Low Volume DoS Attacks with Genetic Programming

Stjepan Picek and Erik Hemberg and Domagoj Jakobovic and Una-May O'Reilly

Abstract We use Genetic Programming in a machine learning approach to learn a detector of DoS-related network intrusion events. We present a one class classifier technique that trains a model from one class of data - normal, i.e., non-intrusion events. Our technique, after ensemble fusion, is competitive with one-class modeling with Support Vector Machines. We compare with three datasets and our best GP-based classifiers are able to outperform one-class SVM. For two out of four test cases, the advantage of GP classifier when compared with one-class SVM is less than 1% which does not represent a significant improvement. On the last two cases, GP achieves significantly better results and making it a viable choice for anomaly detection task.

1 Introduction

Denial of Service (DoS) cyber attacks present a serious threat to computer systems and inflict significant economic damage. They disrupt critical public and enterprise services. DoS attacks can be characterized by their *attack surface*, e.g. application resources, protocol or network, by their *volume*, and in terms of *how they are measured*: bandwidth magnitude is measured in bits per second (Bps), protocol layer attacks are measured in packets per second and application layer attacks are measured in requests per second. Many DoS attacks are advanced by malicious network intrusion events that flood a system's resources so that services to legitimate requests are denied.

One such example is the so-called SYN Flood attack [6]. It exploits a known vulnerability in the TCP (Transfer Connection Protocol) connection sequence. This

Stjepan Picek, MIT, CSAIL, e-mail: stjepan@computer.org · Erik Hemberg MIT, CSAIL, e-mail: erik.hemberg@gmail.com · Domagoj Jakobovic, University of Zagreb, Faculty of Electrical Engineering and Computing e-mail: domagoj.jakobovic@fer.hr · Una-May O'Reilly MIT, CSAIL, e-mail: unamay@csail.mit.edu

sequence has three steps. (1) host A sends a SYN request to open a connection to host B. (2) B then responds with a SYN-ACK response and waits while holding resources for A (3) A confirms with an ACK. In a SYN Flood attack, the requester A sends multiple SYN requests but either does not respond to B's SYN-ACK response, or sends the SYN requests from an IP address that is not its own (*spoofed*). Either way, B continues to wait for each ACK response so eventually no new connections can be made and ultimately denying B from providing connection services. Other DoS flood attacks take advantage of similar ways to tie up a resource. Low volume DoS flood attacks, including SYN Flood, rely on flying under the radar, i.e., sparsely displaying a signature in network traffic flows, in order to evade intrusion detection techniques [12]. In this paper we focus on intrusions closely related to DoS attacks. Detecting such attacks is often very difficult but highly valuable because DoS attacks can be high volume.

One option stemming from the nature-inspired computation area for developing intrusion classifiers is to use Genetic Programming (GP). GP has been used to learn a binary classifier that discriminates between the normal and the anomalous data a number of times with good results [7]. When the problem offers only one class (i.e., normal data) some researchers have approached the problem by synthetically creating a second, "non-normal", class of data and continuing to use GP to learn a binary classifier [5]. Others have approached this problem from the unsupervised learning perspective [13].

Differing from this approach, herein we present the design of a GP intrusion classifier (i.e., a detector) that requires only one class to regress. We train the classifier with only normal data by selecting for mappings with outputs that lie within a restricted range. When we evaluate candidate classifiers, in the testing phase, we test with both normal and anomalous data. Afterward we ensemble selected classifiers from our runs and perform further evaluations. In order to evaluate its efficiency we compare our method with one-class SVM on several datasets (network logs) that are either created for the purpose of benchmarking a classifier or taken from practical settings. Our one-class classification technique is general, i.e., it extends to problems of anomaly outside intrusion detection.

Our main contributions are:

- Design of a new one-class GP classifier using symbolic regression and interval mapping.
- Design and evaluation of ensemble classifiers based on the one-class GP classifier.
- Experimental evaluation of our approach on a number of datasets and comparison with one-class SVM.

The rest of this paper is organized as follows. In Section 2 we present the design of our GP-based one-class classifier and outline our ensemble fusion method. In Section 3 we summarize related work. Section 4 presents the datasets we use, experimental settings, results, and discussion. Finally, in Section 5 we consider possible future research and conclude.

2 Our Method

An adversarial model defines the scope of intrusions a classifier is expected to detect. Our model assumes the adversary is able to launch attacks identifiable as sourced from any IP address and can, prior to intrusion, conduct a reconnaissance phase to identify its target. The adversary can launch different types of attacks, not only DoS which imply it seeks to penetrate the network perimeter of the target. Intrusions outside this model are not considered. One such example is an adversary who can influence the GP training process. We further assume that we only need to collect data from a network that has not experienced intrusion providing us with data of class “normal”.

2.1 Intuition

We ask GP to evolve a mapping that compresses normal class data into a predefined target interval $I = [lower \dots upper]$. An output for an input datum that is anomalous should be mapped outside I. On an intuitive level, we seek a compressive relation that embeds data from a high dimensional feature space to a one-dimensional space with a specific “normal” interval for normal class data and values outside the interval for “anomalous” data. We want the interval to be as small as possible to minimize both false negatives, i.e., anomalous data mapping into the interval, and false positives, i.e., normal data falling outside it.

Since GP computes floating point numbers, we use how floating points are stored in computers and their discretely decreasing density to define the target intervals for a classifier. For floating point representation, computers use a system highly resembling that of exponential notation where a number can be stored in binary notation (recall any number can be expanded to the binary representation) as $\pm n \times 2^E$ with n being the significand, E the exponent, and $1 \leq n < 2$. A 32-bit word can then be divided into 1 bit for the sign, 8 bits for the exponent, and 23 bits for the significand. Since the significand has 23 bits, the gap (precision) between the floating point value 1 and the next larger floating point (in binary, first bit equal to 1, followed by 21 zero bits and the last bit equal to one) is $\eta = 2^{-22}$. That gap between adjacent floating point numbers becomes bigger as the magnitudes of the numbers become bigger, and smaller as the magnitudes of the numbers become smaller.

We illustrate the density of floating point numbers in a small example in Figure 1. The precision as well as the magnitudes of the floating point numbers are defined by the IEEE floating point representation [1]. Additional details about the floating point and how are they stored in computers can be found in [15].

In our approach, we treat the choice of the target interval as a parameter of the learning method and we use intervals $[0,1]$, $[1,2]$, $[2,3]$, $[3,4]$, $[4,5]$, $[7,8]$, $[8,9]$, $[15,16]$, $[16,17]$, $[31,32]$, and $[32,33]$. These intervals are chosen because either the lower or the upper boundary coincides with an increasing power of 2, since the powers of 2 mark the change in the density of floating point representation. We

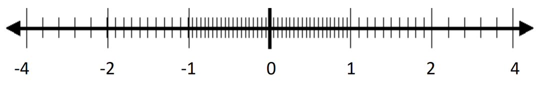


Fig. 1: Density of floating point values on the real line

ask GP to evolve classifiers for smaller and smaller intervals. We expect GP to find larger intervals more easily, e.g. forcing the output for all training instances to $[0,1]$ could be done trivially, but the interval $[16,17]$ requires a mapping with a smaller range. After a sweep through different target intervals, we select a classifier from the smallest interval where we obtain acceptable training performance.

We use absolute values so we consider both signs (negative and positive); range $[0,1>$ (or $<-1,1>$ if not taking absolute value) holds half of all IEEE floating point numbers, which is approximately 2^{63}, while the range $[1,inf>$ holds the other half. Range $[1,2]$ holds approximately 2^{53} numbers, which are all the combinations using the same exponent. Ranges $[2,3]$ and $[3,4]$ both hold 2^{52} numbers, $[4,5]$ and $[7,8]$ hold 2^{51} etc.

2.2 Formal Definition

Our GP evolves a function GPF in the form of a regression tree that produces a single value. Let GPF be a function (classifier) evolved by GP and R the set of intervals we regress our values to, i.e., $R \in [0,1]$, $[1,2]$, $[2,3]$, $[3,4]$, $[4,5]$, $[7,8]$, $[8,9]$, $[15,16]$, $[16,17]$, $[31,32]$, $[32,33]$. Next, n denotes the number of instances in the dataset and X is the vector of m features – $(x_1,\dots,x_m)$. Y is the set of all possible class labels, here $Y=(0,1)$, $y \in Y$ is the actual label of instance Xi, and $\hat{y}$ is the predicted label of X.

Our goal is to learn the classifier $y = f(X_i;R)$ where R denotes the target interval. The predicted label for $\hat{y}$ is "normal" for $\hat{y} = GPFx(X_i;R) \in R$ and "anomaly" for $\hat{y} = GPF(X_i;R) \notin R$. In Table 1 we present how to derive classification outcomes. The normal class is denoted as negative ("0") and anomaly as positive ("1"), but the choice of labels is arbitrary.

Table 1: Classification Outcomes for $\hat{y} = GPF^R(X_i)$

Outcome	Description
TP	$\hat{y} = GPF(X_i;R) \notin R$ and class$(X_i) = 1$
FN	$\hat{y} = GPF(X_i;R) \in R$ and class$(X_i) = 1$
TN	$\hat{y} = GPF(X_i;R) \in R$ and class$(X_i) = 0$
FP	$\hat{y} = GPF(X_i;R) \notin R$ and class$(X_i) = 0$

Note that in the training phase, which includes only normal instances, only true negatives and false positives are possible and we evaluate a model's accuracy as follows:

$$ACC_1(GPF^I) = \frac{TN}{TN+FP}. \tag{1}$$

There are a number of design options to supplement the method's fitness function to express more than accuracy. One option is to posit that all the features of the data are relevant to the unseen "anomalous" class and therefore GP should use all (or at least the majority) of input features. Fitness pressure to express this is achieved by multiplying the classifier accuracy with the *fraction of the features* from the entire input set that appear in the GP model. This should further serve to prevent GP from evolving "cheat" solutions that are trivial mappings that don't depend on the input at all.

The resulting training fitness function for the GP is:

$$fitness = ACC_1 \cdot \frac{\texttt{nUsedFeatures}}{\texttt{nAllFeatures}}, \tag{2}$$

Here, `nUsedFeatures` is the number of features that appear in the tree and `nAllFeatures` is the total number of features in the dataset. With this fitness function, only a solution including all the features can have the best possible fitness. Another design option could maximize the range of the mapped outputs for the training set or the number of unique outputs. Our experiments use the first design option. We check whether some features in the evolved classifier are used in a trivial way, e.g., a feature that is subtracted from itself, and we do not include such features in the `nUsedFeatures` number. Naturally, it is still possible that although all features are used in the tree that some of them are actually canceled out.

After each generation of GP with the training set, we check the best classifier on cross-validation split and stop immediately when the cross-validation score gets worse. We report the accuracy of an evolved classifier using test data distinct from the training or cross-validation splits, composed of both normal and anomaly instances. Our measure is:

$$ACC_2(GPF^I) = \frac{TP+TN}{TP+TN+FN+FP}. \tag{3}$$

We also report the $F1$ measure on the test data:

$$F1 = 2\frac{precision \cdot recall}{precision + recall}, \tag{4}$$

where *precision* is the number of correct positive results divided by the number of all positive results, while recall is the number of correct positive results divided by the number of positive results that should have been returned.

We do not have a feature selection step within our method. While fewer features will make classification faster and may help with training set accuracy we anticipate some features that are not relevant to discriminating "normal" may be relevant for "anomaly". Feature selection only makes sense when learning a binary classifier from this perspective. We present the pseudocode for our GP classifier in Algorithm 1.

Algorithm 1 One-class GP Classifier.

```
Input:
$R$ – set of ranges,
$S_{train}$ – training set,
$S_{xval}$ – cross-validation set,
$S_{test}$ – testing set,
Output:
$GPF^r$ – evolved classifier,
$ACC_1(GPF^r, S_{train})$ – accuracy on the training set for the evolved classifier,
$ACC_2(GPF^r, S_{test})$ – – accuracy on the testing set for the evolved classifier,
$F1(GPF^r, S_{train})$ – F1 measure on the testing set for evolved classifier,
$fitness(GPF^r, S_{train})$ – fitness score for the training set and evolved classifier,
repeat
    $r = next\ in\ R$
    repeat
        $find\ best\ classifier\ that\ maximizes\ fitness\ on\ training\ set$
    until $ACC_1(GPF^r_{current}, S_{xval}) < ACC_1(GPF^r_{previous}, S_{xval})$
until $all\ ranges\ tested$
```

2.3 Ensemble Formation

Since GP produces a population of solutions, and since we execute the GP in several runs, a natural question is whether it is possible to use more than one solution in order to obtain more reliable results. Consequently, we use GP as an ensemble classifier where the size of ensemble varies. We note that GP ensembles were also used before for intrusion detection frameworks but the GP part was understandably different from ours [8].

To construct an ensemble, we simply choose some among the evolved models from multiple runs. The models are included in the ensemble in the order of their decreasing output standard deviation, up to the target ensemble size. To obtain the fused result, we either use voting (where we must use odd number of models) or compute the average output value before determining the instance class. To conclude, in order to use ensemble formation, we need to select the following parameters:

1. The target range selection policy.
2. The model selection policy and the number of classifiers in the ensemble.
3. The prediction fusing policy.

3 Related Work

Anomaly based detection is a well researched topic in the last decade and more with many papers examining various defense types or algorithms to be used. In this section, we give only a short overview of relevant works in order to better understand the variety of approaches used up to now. Intrusion detection techniques are usually divided into **signature based** and **anomaly detection based** approaches. In the signature based approaches one relies on recognizing the signatures of attacks (e.g. hash values that are characteristic for certain attack types). Such detection techniques are easily avoided by modifying the attack or using previously unknown attack (zero-day attack). Anomaly detection systems rely on recognizing what is normal traffic and categorizing all that does not fit the description of normal into anomaly.

One-class GP is an idea introduced by Curry and Heywood where they artificially create the second class (outliers) on the basis of the normal data that is available [5]. We note that we do not consider it to be appropriate for network anomaly detection scenarios since one can only create data that does not belong to the normal data that are available, which does not mean that such created data correspond to anomalies.

Cao et al. experiment with one-class classification by using kernel density function where the density function is approximated by using genetic programming symbolic regression [3]. Their results improve over standard one-class KDE and the authors report good results on a number of datasets where one is the KDD Cup dataset. We construct our one-class GP differently, where we do not artificially create anomaly data. In addition, we do not use feature selection when training the one class GP.

To and Elati developed a one-class GP where they use only one class in the training [22]. In their approach, GP tries to find a curve that fits all patterns in the training set. Next, patterns close to the curve are selected where the proximity is evaluated with Euclidean distance. Then, if an instance belonging to the testing set is close to the trained patterns, it is defined as belonging to the normal class.

Orfila et al. use genetic programming in order to train easy-to-understand network intrusion detection rules [14]. The authors concluded that GP can be used to generate short rules that are easily understood which facilitates the understanding of the semantics of the attack.

Song et al. use genetic programming to detect anomalies in KDD Cup dataset where the authors use the hierarchical dynamic subset selection in order to be able to train on around 500 000 instances [20].

There has been also a series of work concentrating on the feature selection for the anomaly detection, see e.g. [24, 26]. Still we note that it is hard to conduct feature selection with anomaly detection since the important features for the normal class do not necessarily need to be important features for anomaly class.

For a somewhat outdated but extensive overview of computational intelligence methods for usages in the intrusion detection system we refer interested readers to [25]. For an overview of machine learning techniques for intrusion detection we refer readers to [23]. Finally, for a general reference on outlier analysis, we refer readers to [2].

4 Experiments

In this section, we first describe the datasets we use for evaluation, our experimental settings, and the algorithm to which we compare. Then, we present results for one-class classification.

4.1 Datasets

We evaluate with three datasets named KDD, NSL-KDD, and Proprietary. KDD is the oldest and was created from manually generated traffic in a controlled (research) network. The traffic has been criticized as unrealistic [19] and some data is redundant. The NSL-KDD dataset is a revision of KDD to address these problems. Both datasets are in common use as benchmarks so we use them. They also offer examples of multiple attack types which is not always possible to obtain from a single real dataset. The third dataset is proprietary. It consists of only one attack type and normal traffic. This type of attack is also present in the KDD and NSL-KDD datasets.

4.1.1 KDD Cup Dataset

The KDD Cup dataset [21] is extracted from nine weeks of raw TCP dump data for a local-area network (LAN) simulating a typical U.S. Air Force LAN being exposed to multiple attacks. The dump consists of about five million connection records selected for training purposes and around two million connection records for testing purposes. The records are grouped into sequences of TCP packets starting and ending at some well defined times. Each sequence can be labeled as either normal or anomalous. Each sequence has 41 features [21]. Anomalous sequences can be further divided into four classes:

1. **DoS** – denial-of-service attacks.
2. **Probe** – surveillance and other probing attacks.
3. R2L – unauthorized access from a remote machine.
4. U2R – unauthorized access to local superuser privileges.

We use 25 000 instances in the training set and 5 500 in the cross-validation set, all normal. One testing set is comprised of 75 000 instances, normal and anomalous. The anomalous class includes instances of two different attack types: DoS and Probe grouped under the "anomaly" label. In the testing set, 35% of instances belong to the anomaly class. We could also include all instances of every attack type. We call this second KDD dataset KDD^* and it consists of 77 000 instances in the testing set (training and cross-validation sets are the same as for the KDD Cup dataset). For the KDD^* testing set, 38% of instances belong to the anomaly class.

4.1.2 NSL-KDD Dataset

The second dataset is the NSL-KDD which attempts to remedy some of the problems of the KDD Cup dataset [9]. The differences are:

- The dataset does not include redundant records in the train set.
- There are no duplicate records in the proposed test sets.
- The number of selected records from each difficulty level group is inversely proportional to the percentage of records in the original KDD Cup dataset.
- The number of records in the train and test sets are smaller (which enables us to use all the instances).

The details for this dataset are the same as for the KDD Cup dataset (i.e., the same number of features) and we use 10 000 instances in the training, 3 500 in the cross-validation, and 22 000 instances in the testing phase. Testing set has 57% of instances belonging to the anomaly class.

4.1.3 Proprietary Dataset

The last dataset we use for evaluation is a proprietary dataset obtained from an anonymized Internet provider. In order to preserve the confidentiality of data, we report only its basic characteristics. It consists of only 9 200 network logs with 3 000 instances belonging to the normal traffic in the training set and 700 instances belonging to the normal traffic in the cross-validation set. As before, instances are starting and ending at some well defined times in accordance to the rules as defined by the firewall in use. Post-hoc analysis showed the remaining instances to be a type of low intensity DoS attack called Syn Flood, see Figure 2. It is possible that this dataset has instances labeled incorrectly, most likely false positives ("normal" labels for anomalous data). This cannot happen with the manually generated datasets NSL-KDD and KDD. Each record consists of 15 features extracted from the raw data. We do not conduct any feature selection since there is no a priori knowledge on what features are the most important. Proprietary testing set has 31% of instances belonging to the anomaly class.

4.2 Genetic Programming Parameters

The input terminals are the features of the dataset and they are treated as real values. To include the nominal features with discrete values, these have been mapped to the set $\{0, 1, \ldots\}$ containing as many values as the given nominal feature. The rest of the features are used without any transformation.

As functions, we use the standard arithmetic binary operators (+, -, *, / (protected)), as well as the square root function and the branch operator *iflte*, which accepts four arguments, returns the third argument if the first one is less than or equal to the

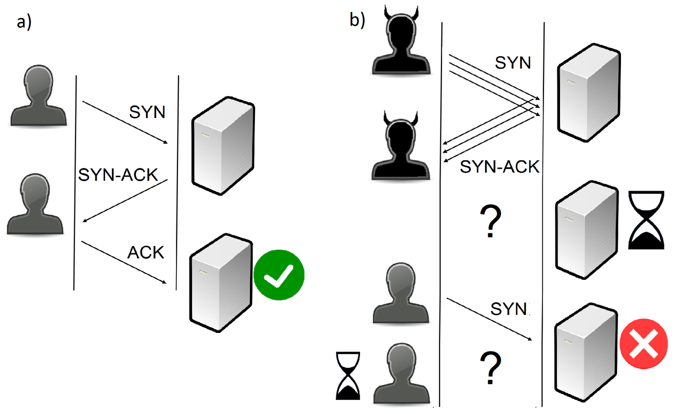


Fig. 2: Syn Flood attack. Part a) Normal traffic between legitimate user and server. Part b) The attacker sends several packets but does not send the "ACK" back to the server. The connections are half-opened and consuming resources on server. A legitimate user tries to connect but the server refuses to open a connection resulting in a denial of service.

second, and the fourth one otherwise. The square root and division operators are protected so that the square returns 0 if the argument is negative, and division returns 1 if the denominator is close to zero.

For all the GP classifiers, the training phase is conducted with a population size of 500 individuals; all the training combinations are executed in 30 runs (repetitions). In the evolution process, GP uses a 3-tournament selection, where the worst of the 3 randomly selected individuals is eliminated. A new individual is immediately created by applying crossover to the remaining two individuals from the tournament. The new individual is then mutated with a probability of 0.5. The crossover is performed with five different tree-based crossover operators selected at random: a simple tree crossover with 90% bias for functional nodes, uniform crossover, size fair, one-point, and context preserving crossover [16]. The mutation operators are subtree, shrink, hoist, permutation, and Gaussian mutation of ephemeral random constants, applied at random for each mutation operation. The GP implementation is based on the Evolutionary computation framework [10].

4.3 Comparison Algorithm

We compare to one-class SVM. SVM is a semi-supervised learning algorithm where the support vector model is trained on instances belonging to only one class [18]. That class is usually called "normal" class and the one-class SVM tries to infer the properties of that class and from them predict which examples are not like the normal class, i.e., they are anomalies. One-class SVM is therefore usually used for anomaly detection due to the fact that the lack of training examples is what defines anomalies. The one-class classification is reached by searching a hyperplane with a maximum margin between the target data and the origin. Note that since SVM decision boundaries are soft, it can be used as an unsupervised algorithm as well. The implementation we use is from LIBSVM [4] as available in the R tool [17]. Further details about one-class classification techniques can be found in [11].

We perform a tuning phase of the SVM parameters for each dataset. In all our experiments we use a radial basis kernel and tune ν and γ parameters. Here, ν parameter is an upper bound on the fraction of margin errors and a lower bound of the fraction of support vectors relative to the total number of training examples. The γ parameter defines how far the influence of a single training example reaches. The parameter values resulting from the tuning phase are given in Table 2.

Table 2: One Class SVM Parameter Tuning

Parameter	KDD/KDD*	NSL-KDD	Proprietary
ν	0.001	0.001	0.5
γ	0.1	0.1	0.001

4.4 Classification Results

In the GP case, we varied the target output interval in the learning phase, and the same interval is used in the testing phase: if an instance in the testing phase is mapped outside the given interval, it is classified as the anomaly. To assess the efficiency of our GP classifier, we consider two measures; accuracy and F1 measure. In this case, the accuracy measure should be considered only as a rough indication of classifier behavior and not as a definitive measure for assessing the performance of a classifier. This is especially correct in scenarios where anomaly data is much less represented than the normal data since then even trivial classifiers would attain a high accuracy by simply putting all measurements in the normal class.

The training results on the KDD Cup dataset for the GP classifier are given in Figure 3a. We observe that the GP easily succeeds in producing models which include all of the features (since the maximum fitness equals to 100%) and still output values in the desired interval. This is especially true for ranges $[0,1], [1,2]$, and $[2,3]$ where it is trivial to fit almost all instances into those ranges. We note that in cases where fitness is lower, this is almost in every instance due to the *accuracy* term in Eq. 2, while the second term equals 1, which means that the GP is able to include all the features easily. As the most interesting ranges we consider $[3,4], [4,5], [15,16], [16,17]$ where the fitness value is still high and behavior is stable.

The testing results are given in Figure 3b for the accuracy and Figure 3c for the F1 measure. While in the training results the models are able to accurately map the training data with absolute precision for some target ranges, we can see from the test results that these ranges do not provide models with generalization capabilities. The results indicate that the most of the evolved classifiers tend to classify all the testing instances as belonging to the normal class; this result is the consequence of forcing the output to the desired range in the training phase. For $[0,1]$ and $[1,2]$ ranges, accuracy is around 65% which actually represents the percentage of normal data in the whole dataset. Consequently, F1 measure reveals the problems with those ranges where we see the results to be 0.

Rather than using the feature information in a meaningful manner, some of the developed models simply "play it safe" and retain the target output value range regardless of the input features. The same result can be observed in the case of the one-class SVM, which also exhibits this behavior, classifying the majority of test instances in the normal class (Table 3). Since the evolved models do not show a great difference in training fitness values, the open question is the identification of model properties that would indicate better performance on the unseen data.

The same models that were trained using the KDD Cup dataset are tested on the KDD^* dataset, which uses the same features. The test results in terms of accuracy and F1 measures are shown in Figures 4a and 4b, respectively. It can be seen that the similar level of accuracy is reached for the same target ranges as in the KDD Cup dataset. Likewise, ranges $[0,1]$ and $[1,2]$ result in trivial classifiers where all data is classified as normal.

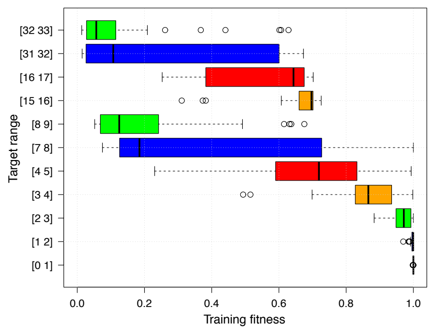
Next, Figures 5a to 5c give results for the NSL-KDD dataset for training, testing with accuracy, and testing with F1 measure, respectively. In the training phase the smallest ranges again easily fit all the data.

The final set of results are given for the proprietary dataset, with training results given in 6a. The testing results are shown in Figure 6b for accuracy and in Figure 6c for the F1 measure. We can observe that in this dataset the test performance over different target ranges is much more diverse, and it is difficult to reach a conclusion of the most effective range.

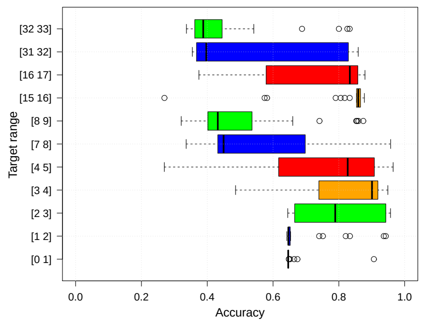
In Table 3 we give results for testing phases for GP as well as for one-class SVM. We note that for all investigated datasets, GP was able to reach higher accuracy and F1 measure than the one-class SVM. We also depict those results in Figures 7a and 7b for accuracy and F1 score, respectively. Still, the GP efficiency varies considerably over multiple training runs, which is an issue we address further in the next section.

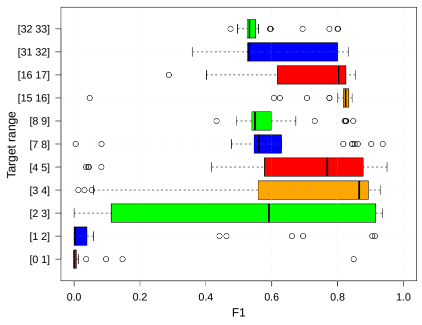
Table 3: Testing results (accuracy/F1)

Algorithm		KDD	KDD*	NSL-KDD	Proprietary
One-class SVM		95.70/94.10	93.70/91.70	80.90/81.20	83.30/73.80
GP $[4,5]$	min	26.94/3.62	27.73/6.82	34.73/16.31	38.87/0.00
	max	96.49/94.99	94.35/92.27	87.83/89.96	99.69/99.50
	median	82.69/76.85	82.20/74.02	71.12/72.27	79.92/75.59



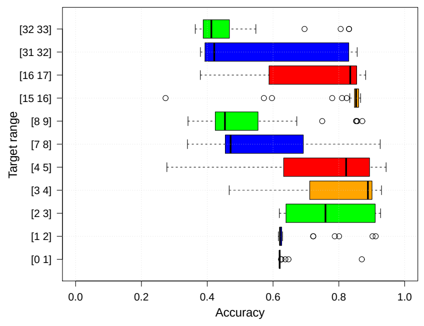
(a) Training results

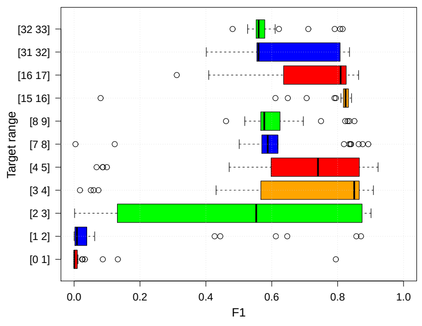




(b) Test results (accuracy) (c) Test results (F1)

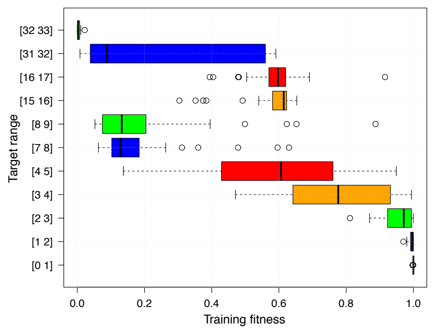
Fig. 3: GP one-class regression, KDD Cup dataset



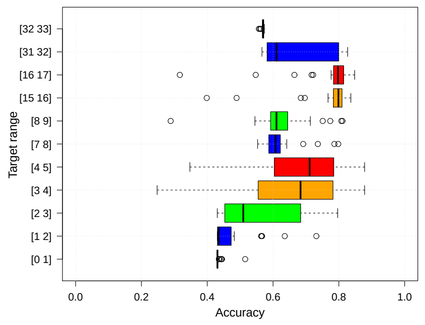


(a) Test results (accuracy) (b) Test results (F1)

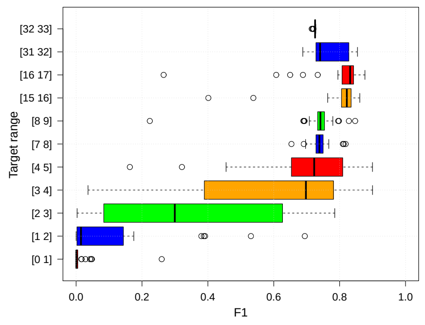
Fig. 4: GP one-class regression, KDD^* dataset



(a) Training results



(b) Test results (accuracy)



(c) Test results (F1)

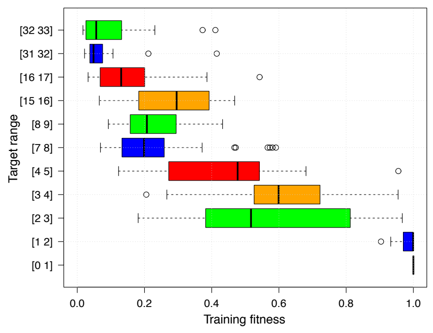
Fig. 5: GP one-class regression, NSL-KDD dataset

Finally, we give an example of a solution (in prefix notation) obtained with GP for one-class classification for the KDD Cup dataset.

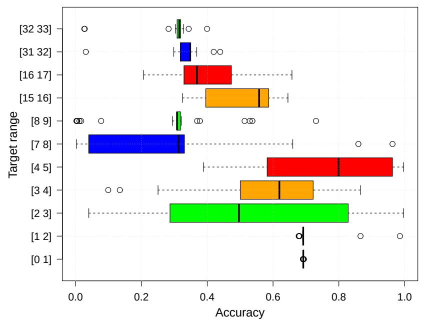
```
iflte((var15+var27)*(var11-var18),iflte(iflte(var10,var2,
var21,var34),iflte(var1,var20,var12,var17),sqrt(var13),
iflte(var16,var3,var14,var30)),iflte(var7,var8,var39,var41)*
(var26-var9),(var28/var27)+(var36*var6))-(var38+var37-
var23/var14-var24/var22)+sqrt(iflte(var25,var15,var37,var16)
+sqrt(var33)-iflte(iflte(var5,var19,var32,var33),var40-var35,
var4*var29,var31*var25))
```

4.5 Selection of GP Models and Ensemble Classifier

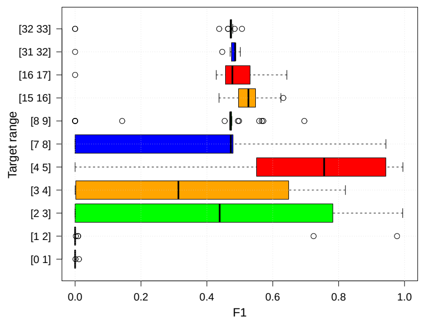
From the results in the previous section we can observe that GP is able to produce highly accurate classifiers. However, there is unfortunately no visible correlation



(a) Training results

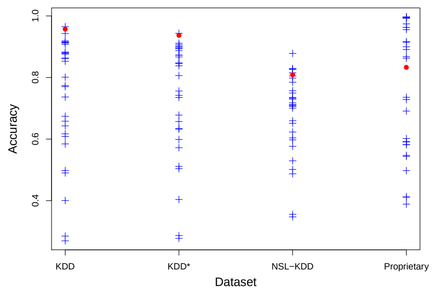


(b) Test results (accuracy)

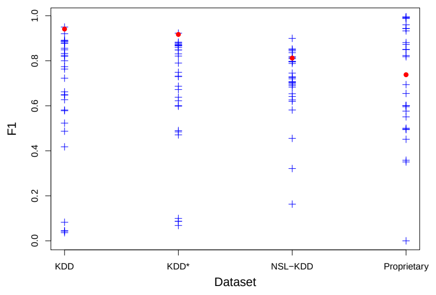


(c) Test results (F1)

Fig. 6: GP one-class regression, proprietary dataset



(a) GP vs one-class SVM (accuracy)



(b) GP vs one-class SVM (F1)

Fig. 7: GP (in blue plus symbol) with target range $[4,5]$ vs one-class SVM (in red filled circle symbol)

between the training fitness of a model and its efficiency on the test dataset, since there are models with high fitness and poor generalization properties. Likewise, a lower training fitness model can still obtain significantly better result on test instances. Therefore, one needs to determine both which target range and which model to use for unseen instances.

As for the target range, we can see from the results in the previous section that ranges below $[4,5]$ should be avoided because the evolved models do not have a satisfactory generalization ability. This is also evident from examining the individual models, which in lower ranges have a very small deviation of output values, or a very small number of unique outputs (e.g., all the instances are mapped to a single output value). These are trivial models which do not actually depend on input data but form an identity function to fit the desired training interval.

For the higher ranges the training fitness decreases as the distance from zero is greater, making it harder to evolve a model that would capture the training instances. Therefore, any as high range as possible that offers high enough training fitness could be considered for use.

To chose a target range relying solely on the training data, we design a selection policy where we use the range that has the largest number of highly fit individuals. In order to select it, we could for instance use the median value or the number of models above some minimal classification score σ where that parameter needs to be user-defined. For all the datasets in the previous section, based on the training results in Figures 3a to 6a, this would result in the range $[4,5]$ as the chosen one; consequently, we will concentrate on this range in further analysis.

As for the choice of a particular model, simply choosing the one with the highest training fitness does not guarantee the most successful performance on the test set. On the contrary, often the models with perfect training score perform poorly on unseen instances. The same behavior is present with the accuracy on the cross-validation data, which also does not show meaningful indication of test performance. Here, the goal would be determining whether the training phase produced a "regular" classifier or just a trivial arithmetic operation to map all the input combinations into the given output interval. One way of performing this analysis is to test the variability of output values, thus forcing the model to include the information in a meaningful way. To achieve this, we analyze the models' behavior on the training set, recording the GP output values for all training instances, and calculate the *standard deviation* of those values. We note that we do not presume any specific distribution of output values, but merely aim to estimate the model efficiency.

We therefore concentrate on all evolved models and sort them not by training fitness, but in the *decreasing* order of standard deviation of the values they produce. A sample of these models for the KDD Cup dataset in the range $[4,5]$ is shown in Table 4. While not perfect, the selection of models based on this criterion (greater values of deviation) provides a more reliable outcome than relying on training fitness only. This simple analysis also immediately reveals a number of models with perfect training score but with deviation close to zero (or exactly zero) which can be quickly discarded.

If we follow the guidelines for the range, we construct the ensembles using the range $[4,5]$; the final results for all datasets are given in Table 5. Regarding the ensemble construction methods, we can immediately note that the averaging method should not be considered; this is the consequence of different models using radically different output values which cannot be added in a meaningful manner.

The obtained results are encouraging since we are able to increase the performance of our GP classifier, where we note that the best results should be obtained with medium sized ensembles (of 7 or 11). At the same time, there is almost no overhead stemming from using GP in ensemble setting. Indeed, since GP works with populations we already have the solutions and the only additional step is sorting them and running the voting or averaging. However, for effective ensembles of target size, the total number of runs should be significantly higher than the ensemble size to provide diversity of models to choose from.

Table 4: Training model selection - KDD Cup, range $[4,5]$

Training fitness	0.6586	0.5587	0.2332	0.4336	0.92584	0.4069	0.66032	0.9042
F1 (test data)	83.7	87.9	3.6	88.6	93.1	88.6	84.3	76.3
St. dev. (training)	$1.8*10^6$	$1.5*10^6$	$1.1*10^6$	200 227	157 564	69 228	59 717	24 795

Table 5: Ensemble test results - all datasets, range $[4,5]$ (F1 measure)

Dataset	KDD		KDD*		NSL-KDD		Proprietary	
Ensemble size	Vote	Average	Vote	Average	Vote	Average	Vote	Average
3	84.5	39.3	83.2	42.8	79.5	63.4	94.3	60.1
7	87.9	59.9	87.4	62.1	85.3	75.1	90.4	57.1
11	91.3	60.2	89.6	62.1	84.4	81.5	65.0	55.3
15	89.7	54.1	88.7	56.2	86.9	78.5	80.0	56.4

5 Conclusion

In this paper, we investigate how to employ genetic programming as one-class classifier in order to detect anomalies in the network traffic. To be able to use GP as one-class classifier we use it in a regression style where all anomaly instances fall out of a specified range. To vary the difficulty of regressing, we use the fact that density of floating point numbers to real values decreases the farther away from zero one goes. The results indicate that the GP is able to cope with the investigated problems and exhibits efficiency comparable to existing state-of-the-art classifiers. We also discuss how to further increase the stability of our classifier by using it in ensemble model.

Since our GP classifier represents a new technique for one-class classification, there are numerous options to follow when considering future work. One direction could be to investigate various density metrics either in post-hoc analysis or already in the fitness function. Next, one extremely interesting option would be to explore explainability. Here, by explainability we mean being able to understand why GP classifies correctly a specific instance. Even more importantly, if GP does not classify an instance correctly, being able to understand why not and what needs to be added to the model in order to classify that instance correctly.

Acknowledgments

This work has been supported in part by Cybersecurity@CSAIL initiative. Additionally, this work has been supported in part by Croatian Science Foundation under the project IP-2014-09-4882.

References

1. IEEE Standard for Floating-Point Arithmetic. IEEE Std 754-2008 pp. 1–70 (2008)
2. Aggarwal, C.C.: Outlier Analysis. Springer Publishing Company, Incorporated (2013)
3. Cao, V.L., Nicolau, M., McDermott, J.: One-Class Classification for Anomaly Detection with Kernel Density Estimation and Genetic Programming. In: Genetic Programming - 19th European Conference, EuroGP 2016, Porto, Portugal, March 30 - April 1, 2016, Proceedings, pp. 3–18 (2016)
4. Chang, C.C., Lin, C.J.: LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology **2**, 27:1–27:27 (2011). Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm
5. Curry, R., Heywood, M.I.: One-Class Genetic Programming. In: Genetic Programming, 12th European Conference, EuroGP 2009, Tübingen, Germany, April 15-17, 2009, Proceedings, pp. 1–12 (2009)
6. Eddy, W.M.: Defenses Against TCP SYN Flooding Attacks - The Internet Protocol Journal - Volume 9, Number 4 (2017). URL http://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-34/syn-flooding-attacks.html
7. Elsayed, S., Sarker, R., Slay, J.: Evaluating the performance of a differential evolution algorithm in anomaly detection. In: 2015 IEEE Congress on Evolutionary Computation (CEC), pp. 2490–2497 (2015)
8. Folino, G., Pizzuti, C., Spezzano, G.: GP Ensemble for Distributed Intrusion Detection Systems. In: S. Singh, M. Singh, C. Apte, P. Perner (eds.) Pattern Recognition and Data Mining: Third International Conference on Advances in Pattern Recognition, ICAPR 2005, Bath, UK, August 22-25, 2005, Proceedings, Part I, pp. 54–62. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
9. Habibi, A., et al.: UNB ISCX NSL-KDD DataSet (2017). URL http://nsl.cs.unb.ca/NSL-KDD/

10. Jakobovic, D., et al.: Evolutionary Computation Framework (2016). URL http://ecf.zemris.fer.hr/
11. Khan, S.S., Madden, M.G.: One-Class Classification: Taxonomy of Study and Review of Techniques. CoRR **abs/1312.0049** (2013). URL http://arxiv.org/abs/1312.0049
12. Kuzmanovic, A., Knightly, E.W.: Low-rate tcp-targeted denial of service attacks: the shrew vs. the mice and elephants. In: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 75–86. ACM (2003)
13. Ni, X., He, D., Chan, S., Ahmad, F.: Network Anomaly Detection Using Unsupervised Feature Selection and Density Peak Clustering. In: M. Manulis, A.R. Sadeghi, S. Schneider (eds.) Applied Cryptography and Network Security: 14th International Conference, ACNS 2016, Guildford, UK, June 19-22, 2016. Proceedings, pp. 212–227. Springer International Publishing, Cham (2016)
14. Orfila, A., Estevez-Tapiador, J.M., Ribagorda, A.: Evolving High-Speed, Easy-to-Understand Network Intrusion Detection Rules with Genetic Programming. In: M. Giacobini, A. Brabazon, S. Cagnoni, G.A. Di Caro, A. Ekárt, A.I. Esparcia-Alcázar, M. Farooq, A. Fink, P. Machado (eds.) Applications of Evolutionary Computing: EvoWorkshops 2009: EvoCOMNET, EvoENVIRONMENT, EvoFIN, EvoGAMES, EvoHOT, EvoIASP, EvoINTERACTION, EvoMUSART, EvoNUM, EvoSTOC, EvoTRANSLOG, Tübingen, Germany, April 15-17, 2009. Proceedings, pp. 93–98. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
15. Overton, M.L.: Numerical Computing with IEEE Floating Point Arithmetic. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2001)
16. Poli, R., Langdon, W.B., McPhee, N.F.: A field guide to genetic programming. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk (2008). (With contributions by J. R. Koza)
17. R Development Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria (2008). URL http://www.R-project.org. ISBN 3-900051-07-0
18. Schölkopf, B., Platt, J.C., Shawe-Taylor, J.C., Smola, A.J., Williamson, R.C.: Estimating the Support of a High-Dimensional Distribution. Neural Comput. **13**(7), 1443–1471 (2001)
19. Shiravi, A., Shiravi, H., Tavallaee, M., Ghorbani, A.A.: Toward Developing a Systematic Approach to Generate Benchmark Datasets for Intrusion Detection. Comput. Secur. **31**(3), 357–374 (2012)
20. Song, D., Heywood, M.I., Zincir-Heywood, A.N.: Training genetic programming on half a million patterns: an example from anomaly detection. IEEE Trans. Evolutionary Computation **9**(3), 225–239 (2005)
21. Tavallaee, M., Bagheri, E., Lu, W., Ghorbani, A.A.: A Detailed Analysis of the KDD CUP 99 Data Set. In: Proceedings of the Second IEEE International Conference on Computational Intelligence for Security and Defense Applications, CISDA'09, pp. 53–58. IEEE Press, Piscataway, NJ, USA (2009)
22. To, C., Elati, M.: A Parallel Genetic Programming for Single Class Classification. In: Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation, GECCO '13 Companion, pp. 1579–1586. ACM, New York, NY, USA (2013)
23. Tsai, C.F., Hsu, Y.F., Lin, C.Y., Lin, W.Y.: Intrusion detection by machine learning: A review. Expert Systems with Applications **36**(10), 11,994 – 12,000 (2009)
24. Wang, W., Gombault, S., Guyet, T.: Towards Fast Detecting Intrusions: Using Key Attributes of Network Traffic. In: Proceedings of the 2008 The Third International Conference on Internet Monitoring and Protection, ICIMP '08, pp. 86–91. IEEE Computer Society, Washington, DC, USA (2008)
25. Wu, S.X., Banzhaf, W.: Review: The Use of Computational Intelligence in Intrusion Detection Systems: A Review. Appl. Soft Comput. **10**(1), 1–35 (2010)
26. Zargari, S., Voorhis, D.: Feature Selection in the Corrected KDD-dataset. In: 2012 Third International Conference on Emerging Intelligent Data and Web Technologies, pp. 174–180 (2012)