Automatic Feature Construction for Network Intrusion Detection

Binh Tran[1], Stjepan Picek[2], and Bing Xue[1(✉)]

[1] School of Engineering and Computer Science, Victoria University of Wellington, 600, Wellington 6140, New Zealand
{binh.tran,bing.xue}@ecs.vuw.ac.nz
[2] Cyber Security Research Group, Delft University of Technology, Mekelweg 2, Delft, The Netherlands
stjepan@computer.org

Abstract. The notion of cyberspace became impossible to separate from the notions of cyber threat and cyberattack. Since cyberattacks are getting easier to run, they are also becoming more serious threats from the economic damage perspective. Consequently, we are evident of a continuous adversarial relationship between the attackers trying to mount as powerful as possible attacks and defenders trying to stop the attackers in their goals. To defend against such attacks, defenders have at their disposal a plethora of techniques but they are often falling behind the attackers due to the fact that they need to protect the whole system while the attacker needs to find only a single weakness to exploit. In this paper, we consider one type of a cyberattack – network intrusion – and investigate how to use feature construction via genetic programming in order to improve the intrusion detection accuracy. The obtained results show that feature construction offers improvements in a number of tested scenarios and therefore should be considered as an important step in defense efforts. Such improvements are especially apparent in scenario with the highly unbalanced data, which also represents the most interesting case from the defensive perspective.

1 Introduction

From its inception, the word cyberspace is used to represent an umbrella of phenomena occurring in computer networks. Besides the advantages stemming from the better connectivity and reachability, cyberspace also has its drawbacks and dangers, where cyberattacks represent one that cannot be neglected. Here, by cyberattacks we consider all deliberate exploitations of computer systems, since such attacks seem to be becoming easier to conduct while their severity grows. The proliferation of such threats on computers, mobile phones, etc. made us all beware that the connectivity we often take for a granted comes with a high price. As an example of that tendency, we mention a fact that NATO suffered from 500 cyber attacks every month in 2016, which represents an increase of 60% from 2015 [1]. One could postulate that the increase and severity of attacks is

Y. Shi et al. (Eds.): SEAL 2017, LNCS 10593, pp. 569–580, 2017.
https://doi.org/10.1007/978-3-319-68759-9_46

evident only for large organizations, but that is not necessarily true. It is enough to consider the 2017 Wannacry ransomware case, where in only one day, more than 230 000 computers in more than 150 countries were infected [2]. Finally, the threats are not limited to only one type of hosts - the Wannacry example spreads over computers running Microsoft Windows operating system, while even more recent threat "Cloak and Dagger" was identified for the Android operating system [3]. We note there are also different types of attacks with respect to the attacker's goals. As an example, the attacker can conduct only a simple network scanning in order to detect some possible system weaknesses to be used later. Alternatively, he can launch massive Distributed Denial-of-Service (DDos) attacks with a goal of making network resources unavailable to its intended users by temporarily or indefinitely disrupting services of a host connected to the Internet.

To fight against such varying threats, there are numerous options one can consider. In the rest of this paper, we limit our attention to one type of threat – network intrusion attacks and consequently defenses against such attacks. A network intrusion attack is an activity, where a malicious user tries to make some sort of unauthorized activity on a computer network. Often, the first defense against it are network intrusion detection techniques. Intrusion detection techniques are usually divided into signature based and anomaly detection based approaches [4]. In the signature based approaches one relies on recognizing the signatures of attacks (for example, hash values that are characteristic for certain attack types). Such detection techniques are easily avoided by modifying the attack or using previously unknown attack (i.e., 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.

When discussing anomaly detection systems, a great deal of research has been invested in utilizing machine learning and evolutionary computation techniques [5,6]. As far as we are aware, up to now there is no research considering how to use evolutionary algorithms to construct higher level features that can result in better classification accuracy for the network intrusion detection. We aim to fill this gap by providing the first results in several relevant scenarios and proposing further research directions.

Genetic programming (GP) is an evolutionary computation technique that can automatically evolve solutions based on the idea of the survival of the fittest. With a flexible representation such as trees and any kind of operators or functions to represent the model, GP is an excellent choice for feature construction. Features constructed by GP have been shown to obtain better discriminating power than the original features [7–9]. In this study, we investigate GP performance in constructing high-level features for the network intrusion detection. The constructed features are expected to improve the classification performance of common learning algorithms when compared with the original feature sets.

The area of network intrusion detection has several specificities which makes it a challenging scenario to consider. The first problem stems from the lack of good datasets since they are in most cases artificially constructed and difficult to

correlate with the realistic scenarios. The second difficulty comes from the fact the classification accuracy is relatively high when compared to other problems and consequently possible improvements obtainable with feature construction are much smaller. Still, due to the importance of the highly-accurate intrusion detection systems and possible damages when anomalies are not detected, even small improvements in accuracy can be of extreme importance. Finally, being able to automatically construct features is beneficial since feature construction is most often done by human experts, which is far from being error prone and is a time consuming process.

The rest of this paper is organized in the following way. In Sect. 2 we discuss the attacker capabilities as well as the defender's goals. We also give details about the dataset we investigate. Section 3 gives a short overview of related work. In Sect. 4 we provide details about GP that is used for feature construction, an brief description of the two classification algorithms used in this paper. Section 5 gives experimental results for several intrusion detection scenarios and a comparison between the constructed features and the original ones. We also provide a discussion on results and remarks on possible future research directions. Finally, in Sect. 6 we give a brief conclusion and future work.

2 Background

2.1 Stages of a Cyberattack

When discussing cyber threats, it is important to realize that often they do not represent atomic actions leading to a success but are rather a succession of steps one needs to do in order to successfully attack a system. Although there are various understandings and attack stage granularities (usually ranging from five to ten stages), some common ones are defined in [10]. An attack usually starts with the reconnaissance phase, where the attacker's goal is to obtain knowledge about the system and find possible weaknesses. Next, incursion follows where the attacker actually penetrates the system. Within the system, the attacker moves laterally in order to discover and exploit the system. Finally, in the exfiltration phase the attacker tries to steal the sensitive data and leave the system undiscovered. We depict five stages of a cyberattack with an emphasis on attacker's goals in Fig. 1.

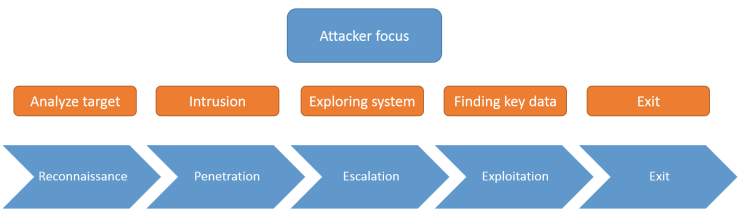

Fig. 1. Five stages of a cyberattack.

We aim to defend against a powerful attacker that has an unbounded number of IP addresses on his disposal, which enables him to run anything from a simple network scanning attacks up to massive DDoS attacks. We do not limit the attacker to be present in any specific stage of a cyber attack. In order to be able to recognize such wide set of threats, we train anomaly detectors for all available attack types as well as for the normal traffic. We assume that the attacker can mount already known attacks as well as the zero-day attacks. Finally, we assume that the attacker cannot interfere with the running of GP or classifiers.

Due to the fact that our system is not fast enough to work in an online setting (where as the new data instances are acquired, GP is able to update the set of constructed features) we consider here only the offline setting, where a number of training instances is acquired and on the basis of them, GP constructs features. Since we are not time-constrained, it is reasonable to assume that we can run additional classifiers on the obtained data in order to classify it into normal or anomaly data. Therefore, we work here only in the binary setting where we have both normal and anomaly data (regardless whether there are different types of anomalies or only one).

Anomaly detection is often considered in the one-class classification setting, where we assume not to have anomaly data at the time of the model training. Still, adding additional classifiers (i.e., building ensemble classifiers) as combinations among one-class, binary, and multi-class classifiers is an option that improves the accuracy of the intrusion detection system. Although it could be claimed that first running feature selection would enable GP to be faster and to work with only the most informative features, we use all available features and expect from GP to recognize the most important ones.

2.2 Datasets

All our experiments are based on the NSL-KDD dataset [11]. This dataset is on the other hand based on one older dataset called the KDD Cup dataset [12]. Here, we first briefly describe the KDD Cup dataset and then we discuss the differences between it and the NSL-KDD dataset.

The KDD Cup dataset is comprised from nine weeks of raw TCP dump data for a local-area network (LAN) simulating a typical U.S. Air Force LAN that was exposed to multiple attacks. Each instance is a sequence of TCP packets starting and ending at some well-defined times. Each record can be labeled into either normal or anomaly data, where anomaly data can be divided into four classes: DoS (denial-of-service attacks), Probe (surveillance and other probing attacks), R2L (unauthorized access from a remote machine), and U2R (unauthorized access to local superuser privileges).

During the years, researchers noticed a number of problems with the KDD Cup dataset. Some of the often mentioned problems are the dissimilarity from the real traffic and issues encountered due to the synthetic data generation and insertion [13]. Still, due to the artificial data creation, one advantage is that we can be sure the data to be correctly classified. Consider a "realistic" dataset where the only proof of an attack is to have some classifier reporting it. In the

case that classifier misclassified, we would propagate errors to the next level of experiments – feature construction. Finally, the KDD Cup dataset offers a wide variety of attacks, which is not something one necessarily encounters in "realistic" scenarios. The improvements of NSL-KDD over the KDD Cup dataset 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.

3 Related Work

Anomaly based detection is a well-researched topic in the last decade or more with many papers examining various defense types and algorithms to be used.

When using GP for anomaly detection, there is a line of works exploring binary and one-class classification. One-class GP is a technique introduced by Curry and Heywood [14] where they artificially create the second class (outliers) on the basis of the normal data that is available. Cao et al. [15] experiment with one-class classification by using kernel density function, where the density function is approximated by using GP symbolic regression. To and Elati [16] developed a one-class GP where they use only one class in the training. In their approach, GP tries to find a curve that fits all patterns in the training set and if an instance belonging to the testing set is close to the trained patterns, it is defined as belonging to the normal class. Song et al. use GP to detect anomalies in the KDD Cup dataset where the authors use hierarchical dynamic subset selection in order to train around 500 000 instances [17].

When considering feature selection, Wang et al. [18] experiment with several methods to find the most informative features for each of the four anomaly classes in the KDD Cup dataset. Zargari and Voorhis [19] conduct feature selection for the NSL-KDD dataset and suggest the most informative features for all anomaly classes.

When considering automatic feature construction for network intrusion detection, the works are sparse and do not consider genetic programming (or any kind of evolutionary algorithms). As an example of automatic feature construction technique, we mention the work by Lee and Stolfo [20], where the authors use several data mining algorithms to build a framework to construct features and models for intrusion detection.

4 Methodology

4.1 Classification Algorithms

We use two simple classifier techniques in order to investigate the performance of the constructed features. We decided to use these classifiers since they are fast and able to reach high accuracies. Note that using more powerful classifiers could improve the results but with a price of a longer classification process.

Naive Bayes (NB). NB classifier is a method based on the Bayesian rule that works under the simplified assumption that the predictor attributes (measurements) are conditionally independent among the features given the target class. Detailed information about the Naive Bayes algorithm can be found in [21].

C4.5. C4.5 is a divide-and-conquer algorithm that split features at tree nodes using the information gain ratio criterion [22]. The node splits in further branches if more information is gained (as measured by the gain ratio) by the split than by keeping all the instances at the node. The trees are first grown to full length and pruned afterwards in order to avoid data overfitting.

With the C4.5 algorithm, we investigate the influence of the confidence factor parameter c that is used for pruning, where smaller values relate to more pruning. We tested that parameter in the range $[0.05, 0.5]$ with a step of 0.05, where we conducted a separate tuning phase for each scenario. Due to the lack of space, we do not show the full tuning results but only the best obtained solutions. For normalVSanomaly we use $c = 0.3$, for normalVSdos $c = 0.35$, for normalVSprobe $c = 0.5$, for normalVSr2l $c = 0.5$, and for normalVSu2r $c = 0.2$.

4.2 Genetic Programming

We use a standard GP algorithm to construct one high-level feature using the single-tree representation as in [8,23]. GP works by maintaining a population of individuals, each of which represents a constructed feature.

To evaluate the goodness of a constructed feature, the evaluation procedure follows an embedded approach, where each individual is evaluated based on its classification performance on the training set. In other words, each individual or the corresponding constructed feature can also be considered as a classifier, which classifies an instance x by executing the following rule:

$$IF \;\; constructed\, F <= 0 \;\; THEN \;\; x \in class_0; \quad ELSE \;\; x \in class_1. \quad (1)$$

Since the network attack detection datasets are usually unbalanced, we use the balanced accuracy [24] as shown in Eq. (2) for fitness evaluation. It is an average of true positive rate (TPR) and true negative rate (TNR), where TPR (or TNR) is the proportion of correctly identified instances of positive (or negative) class.

$$fitness = \frac{1}{2}\left(TPR + TNR\right), \quad (2)$$

In order to avoid underfitting as well as overfitting, we use a dynamic stopping criterion. During the evolutionary process, if the validation accuracy does not improve after 50 generations, GP stops and returns the model with the best validation accuracy; otherwise, GP continues to run until a maximum number of generations is reached. After the evolutionary process, the GP individual having the best validation accuracy is returned as the best solution. Algorithm 1 shows the pseudo code of the GP-based feature construction method which returns a new constructed feature.

Algorithm 1. GP-based feature construction method

Input : Training_data
Output: The best constructed feature

begin
  Training data is equally split into *train_set* and *valid_set*;
  Initialize a population of GP individuals/trees;
  not_improved ← 0;
  while *Maximum iterations is not reached or the best solution is not found or not_improved* < 50 **do**
    for $i = 1$ *to Population Size* **do**
      $transf\_train$ ← Calculate constructed feature of individual i on *train_set* ($transf\_train$ has only one feature, i.e. the constructed feature) ;
      $fitness$ ← classification accuracy of $transf\_train$ using Rule 1 and Eq. 2;
    end
    best_gen_ind ← individual that have the highest $fitness$;
    $transf\_valid$ ← Calculate constructed feature of the *best_gen_ind* on *valid_set* ($transf\_valid$ has only one new feature) ;
    valid_acc ← classification accuracy of $transf\_valid$ using Rule 1 ;
    if *(valid_acc is improved)* **then**
      Update *valid_acc* ;
      not_improved ← 0;
    else
      $not\_improved++$;
    end
    Select parent individuals using tournament for breeding;
    Create offspring individuals by applying crossover or mutation on the selected parents;
    Place new individuals into the population of the next generation;
  end
  best_ind ← individual that have the highest *valid_acc*;
  Return the *best_ind*;
end

Table 1. Datasets

Dataset	#Features	#Instances	Class-distribution
normalVSanomaly	40	47 736	48.52%–51.48%
normalVSdos	39	39 852	58.12%–41.88%
normalVSprobe	38	27 870	83.10%–16.90%
normalVSr2l	40	26 123	88.66%–11.34%
normalVSu2r	40	23 371	99.10%–0.90%

4.3 Dataset Details

To test the performance of our method, five binary class datasets are generated from the NSL-KDD dataset. To obtain the instances, we take 10% of all measurements (which are already labeled) and we create datasets by combining instances belonging to the Normal class with instances belonging to each of the four anomaly classes.

Finally, we combine Normal instances with all anomaly instances (belonging to all 4 classes) and simply assigning them to the anomaly class (i.e., the binary setting). After generating these datasets, we remove those features that have only

a single value since it does not provide any usefuly information. The number of remaining features of each dataset is shown in Table 1. As it can be seen from the class distribution, three out of the five datasets are extremely unbalanced (normalVSu2r dataset has less than 1% of instances belonging to one class), which makes these problems more challenging for machine learning algorithms. Each dataset is stratified and equally split into three data subsets for training, validation, and testing. These subsets are standardized based on the training set before feeding it into GP for feature construction.

5 Experiments

5.1 Experiment Configuration and Parameter Settings

To test the performance of GP in constructing better discriminating features, we compare the performance of common classification algorithms including Naive Bayes and Decision Tree using the resulting feature sets and the original feature set. As GP is a stochastic algorithm, 30 independent runs with different random seeds are applied on each training set. The resulting feature sets are tested on the test set. To eliminate the statistical variations, all comparisons are done on the 30 test results using the Wilcoxon significance test.

Table 2 describes the parameter settings for GP. The function set comprises of 7 functions, 5 of which are arithmetic operators (addition, subtraction, multiplication, square root, and protected division). Function max returns the maximum values from the two inputs and if returns the second argument if the first argument is positive and returns the third argument otherwise. The terminal set comprises of all the original features. ECJ package [25] was used to implement the system. We used the standard crossover operator provided by the package, which swaps two randomly picked subtrees of two parents, and the standard mutation, which replaces a randomly picked subtree of the parent by a new subtree.

Table 2. GP parameters and experiment settings

Function set	$+, -, \times, /, \sqrt{}, max, if$
Initial population	Ramped half-and half
Maximum tree depth	10
Initial maximum tree depth	3
Generations	500
Population size	1 024
Crossover rate	0.4
Mutation rate	0.6
Elitism size	10
Selection method	Tournament method
Tournament size	7

5.2 Results and Analysis

Table 3 shows the test accuracy of NB and DT using the resulting constructed feature sets compared with the "Full" (i.e., using the original feature set) feature set. For each classification algorithm, the best (B), the average, and the standard deviation (A ± Std) over the 30 independent runs are reported. In this column, we also display the Wilcoxon significance test results of the corresponding feature set over Full feature set with significance level of 0.05. "+" or "–" means that the result is significantly better or worse than the Full set and "=" means that their results are similar. In other words, the more "+", the better the resulting feature sets. The time (in minutes) used to train the constructed features is shown under the dataset name. Column "#F" shows the average size of each feature set.

We can observe that the running time is correlated with the sample size of the dataset. Note that although the time to train the constructed features ranges from 38 to 92 min for each dataset, it is only run once and can be offline. After the constructed features are learned, the execution time of the tree to produce the new features is negligible.

Table 3. Test results using NB and DT

Dataset	Features	#F	B-NB	A ± Std-NB	B-DT	A ± Std-DT
normalVSanomaly	Full	40.00	84.40	84.40 ± 0.00	98.44	98.44 ± 0.00
92.21 (m)	FullCF	41.00	87.31	84.71 ± 0.84 =	98.53	98.37 ± 0.09 –
normalVSdos	Full	39.00	91.90	91.90 ± 0.00	99.83	99.83 ± 0.00
77.61 (m)	FullCF	40.00	95.62	92.50 ± 0.90 +	99.84	99.75 ± 0.06 –
normalVSprobe	Full	38.00	93.10	93.10 ± 0.00	99.17	99.17 ± 0.00
51.97 (m)	FullCF	39.00	95.27	93.62 ± 0.64 +	99.40	99.16 ± 0.09 =
normalVSr2l	Full	40.00	85.25	85.25 ± 0.00	96.10	96.10 ± 0.00
50.84 (m)	FullCF	41.00	89.83	87.00 ± 2.09 +	96.95	96.27 ± 0.37 +
normalVSu2r	Full	40.00	91.03	91.03 ± 0.00	89.27	89.27 ± 0.00
38.98 (m)	FullCF	41.00	92.36	91.06 ± 0.60 =	95.67	92.06 ± 1.79 +

As can be seen from Table 3 that feature sets with added one constructed feature (FullCF) are able to outperform the original sets in all considered cases. There, when using the Naive Bayes classifier, the difference is relatively large, while when using Decision Tree classifier, the difference in the accuracy is less pronounced. The FullCF helps NB achieve significantly better results in three out of five cases. The remaining two datasets, namely normalVSanomaly and normalVSu2r, obtain a similar accuracy. We also notice that the best accuracy that NB achieves using FullCF is always higher than using Full set. Using FullCF and considering averaged values, DT performance is better on 2 datasets, similar on 1 and worse on the remaining 2. We emphasize that bigger differences can be seen in the last three datasets, where the imbalance between the classes

is more significant. Note the last scenario – normalVSu2r where the accuracy improvement is more than 6% and the class distribution is 99.10%–0.90%.

To better assess the influence of the added constructed feature, next we present the testing results for scenarios, where we use only one feature (or GP tree) – "CF" set as a binary classifier. As can be seen from Table 4, using one constructed feature results on average in performance degradation for DT on the first three datasets when compared with the original or FullCF feature set. Still, considering the fact that we use only one feature, the results are promising for scenarios, where extremely fast classification is needed. On the other hand, in scenarios with highly unbalanced data as in the last two datasets, this binary classifier with only one constructed feature still achieves higher accuracy than DT using Full or FullCF feature set.

Table 4. Testing results, GP using CF, accuracy. 1 – normalVSanomaly, 2 – normalVSdos, 3 – normalVSprobe, 4 – normalVSr2l, 5 – normalVSu2r

Statistics	1	2	3	4	5
Best	97.13	99.64	99.08	97.28	96.82
Average	96.26	99.35	98.29	96.36	95.03
StdDev	0.75	0.21	0.34	0.48	0.92

Finally, in Table 5 we display the testing results for the FullCF scenario. Note that the results are averaged over 30 runs and we use the C4.5 parameters obtained after a tuning phase for each scenario. The tuning phase is done on a single (for each scenario) training set where we select those uniformly at random. Notice that the first three scenarios give similar accuracies as those where the tuning phase was done on Full feature set (see Table 3) but in the last two scenarios the improvement is even more significant (considering Best values) with the tuning done on FullCF feature set.

Table 5. Testing results, DT using FullCF, accuracy. 1 – normalVSanomaly, 2 – normalVSdos, 3 – normalVSprobe, 4 – normalVSr2l, 5 – normalVSu2r

Statistics	1	2	3	4	5
Best	98.32	99.78	99.96	98.62	99.76
Avg	94.95	98.98	99.14	96.66	99.05
StdDev	3.61	2.85	0.19	3.91	0.48

Since our work represents only a start of investigations on GP feature construction for intrusion detection, there are many possible research directions one could follow. We briefly discuss only three of those. An obvious continuation of this work would be to consider more complex classifiers like Support Vector

Machines. Naturally, this would come with an added cost in the evolutionary process. The second interesting option would be to investigate how to reduce the false positive rate. This could be done by incorporating appropriate term in the fitness function for GP. Finally, here we discuss binary setting but we believe it would be of extreme importance to consider one-class classification and GP feature construction. In such a scenario, one would build new features that better represent normal class and hopefully help to further discriminate between normal and anomaly data.

6 Conclusions

In this paper, we consider the task of automated construction of higher-level features via GP for the network intrusion detection problem. Although this work should be considered only as a preliminary investigation in that direction, our results show that constructed features are able to increase the accuracy especially in scenarios where we observe highly unbalanced data. Since such unbalanced data represents a usual scenario for network intrusion detection (where we have either many normal instances and only sparse anomalies as in the reconnaissance phase or where most of the instances belong to the anomaly class like in DDoS attack) the proposed can be considered highly beneficial. In the future, we will consider other scenarios and datasets to further examine the performance of the proposed algorithm. We also intend to develop new approaches to further investigate the potential of feature construction on solving complex network intrusion problems.

References

1. Browne, R.: Nato: we ward off 500 cyberattacks each month, January 2017. http://edition.cnn.com/2017/01/19/politics/nato-500-cyberattacks-monthly/
2. Symantec: Ransom.wannacry, March 2017. https://www.symantec.com/security_response/writeup.jsp?docid=2017-051310-3522-99
3. Fratantonio, Y., Qian, C., Chung, S., Lee, W.: Cloak and Dagger: from two permissions to complete control of the UI feedback loop. In: Proceedings of the IEEE Symposium on Security and Privacy, Oakland, San Jose, CA, May 2017
4. García-Teodoro, P., Díaz-Verdejo, J., Maciá-Fernández, G., Vázquez, E.: Anomaly-based network intrusion detection: techniques. Syst. Chall. Comput. Secur. **28**(1–2), 18–28 (2009)
5. 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)
6. Tsai, C.F., Hsu, Y.F., Lin, C.Y., Lin, W.Y.: Intrusion detection by machine learning: a review. Expert Syst. Appl. **36**(10), 11994–12000 (2009)
7. Al-Sahaf, H., Al-Sahaf, A., Xue, B., Johnston, M., Zhang, M.: Automatically evolving rotation-invariant texture image descriptors by genetic programming. IEEE Trans. Evol. Comput. **21**(1), 83–101 (2017)
8. Tran, B., Xue, B., Zhang, M.: Genetic programming for feature construction and selection in classification on high-dimensional data. Memet. Comput. **8**(1), 3–15 (2015)

9. Tran, B., Zhang, M., Xue, B.: Multiple feature construction in classification on high-dimensional data using GP. In: IEEE Symposium Series on Computational Intelligence (SSCI), pp. 210–218, December 2017
10. Symantec: preparing for a cyber attack, January 2017. http://www.symantec.com/content/en/us/enterprise/other_resources/b-preparing-for-a-cyber-attack-interactive-SYM285k_050913.pdf
11. Habibi, A., et al.: UNB ISCX NSL-KDD dataset. http://nsl.cs.unb.ca/NSL-KDD/
12. 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 2009, Piscataway, NJ, USA, pp. 53–58. IEEE Press (2009)
13. 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)
14. Curry, R., Heywood, M.I.: One-class genetic programming. In: Vanneschi, L., Gustafson, S., Moraglio, A., De Falco, I., Ebner, M. (eds.) EuroGP 2009. LNCS, vol. 5481, pp. 1–12. Springer, Heidelberg (2009). doi:10.1007/978-3-642-01181-8_1
15. Cao, V.L., Nicolau, M., McDermott, J.: One-class classification for anomaly detection with kernel density estimation and genetic programming. In: Heywood, M.I., McDermott, J., Castelli, M., Costa, E., Sim, K. (eds.) EuroGP 2016. LNCS, vol. 9594, pp. 3–18. Springer, Cham (2016). doi:10.1007/978-3-319-30668-1_1
16. 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 2013 Companion, pp. 1579–1586. ACM, New York (2013)
17. Song, D., Heywood, M.I., Zincir-Heywood, A.N.: Training genetic programming on half a million patterns: an example from anomaly detection. IEEE Trans. Evol. Comput. **9**(3), 225–239 (2005)
18. 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 2008, pp. 86–91. IEEE Computer Society, Washington, DC (2008)
19. 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, September 2012
20. Lee, W., Stolfo, S.J.: A framework for constructing features and models for intrusion detection systems. ACM Trans. Inf. Syst. Secur. **3**(4), 227–261 (2000)
21. Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian network classifiers. Mach. Learn. **29**(2), 131–163 (1997)
22. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc., San Francisco (1993)
23. Tran, B., Xue, B., Zhang, M.: Using feature clustering for GP-based feature construction on high-dimensional data. In: McDermott, J., Castelli, M., Sekanina, L., Haasdijk, E., García-Sánchez, P. (eds.) EuroGP 2017. LNCS, vol. 10196, pp. 210–226. Springer, Cham (2017). doi:10.1007/978-3-319-55696-3_14
24. Bhowan, U., Johnston, M., Zhang, M., Yao, X.: Reusing genetic programming for ensemble selection in classification of unbalanced data. IEEE Trans. Evol. Comput. **18**(6), 893–908 (2014)
25. Evolutionary Computation Laboratory: ECJ: a Java-based evolutionary computation research system. https://cs.gmu.edu/eclab/projects/ecj/