Genetic Programming Feature Extraction with Different Robust Classifiers for Network Intrusion Detection

Khaled Badran
Military Technical College
Cairo

Alaa Rohim
Military Technical College
Cairo

ABSTRACT
In this paper, we compare the performance of three traditional robust classifiers (Neural Networks, Support Vector Machines, and Decision Trees) with and without utilizing multi-objective genetic programming in the feature extraction phase. This work argues that effective feature extraction can significantly enhance the performance of these classifiers. We have applied these three classifiers stand alone to real world five datasets from the UCI machine learning database and also to network intrusion "KDD-99 cup" dataset. Then, the experiments were repeated by adding the feature extraction phase. The results of the two approaches are compared and conclude that the effective method is to evolve optimal feature extractors that transform input pattern space into a decision space in which the performance of traditional robust classifiers can be enhanced.

General Terms
Pattern Recognition, Classification, Network Intrusion.

Keywords
Genetic Programming, Feature Extraction, Neural Network, Support Vector Machines, Decision Tress.

1. INTRODUCTION
In machine learning classification applications feature extraction phase starts from an initial set of measured data with the aim to derive a new values (features) that enhance learning and generalization steps while building the classifier, and in some cases leads to better human interpretations. While feature selection is to choose the optimum features with the aim to reduce the redundant and irrelevant information to the output classes from the complex input data [1].

Feature extraction and selection both can reduce the dimensionality of the data, simplify the learned model in which it can increase the performance of the classifier and generalize better on unseen data. The difference between the two stages is that selection occurs among the original features while extraction occurs after mapping the input space into new decisions space using the derived mapped features. Thus the feature extraction can be regarded as finding an optimal sequence of operations subject to some criterion.

Feature extraction is a challenging issue as it is considered as a search problem. Where the search space is the possible feature compositions and subsets. However, this search needs to be domain independent without utilizing a prior domain expert knowledge. A variety of search techniques have been applied to feature selection, such as complete search, greedy search, heuristic search, and random search [2]-[5] However, these methods still suffer from high complexity and to get trapped in local minimum.

Recently, evolutionary computation (EC) as a global heuristic search techniques and in specific genetic programming been employed in feature extraction process Genetic Programming employs complex pattern representations such as trees to automatically generate a reasonably small yet effective number of features by combining the basic building blocks which are typically used to construct features, and evaluating their fitness automatically. The application of GP to classification offers some interesting advantages, the main one being its flexibility, which allows the technique to be adapted to the needs of each particular problem. For example, any kind of suitable operation or function can be used inside that representation

Genetic programming is an evolutionary technique used to create computer programs that represent approximate or exact solutions to a problem [6]. Based on the Evolution Theory of Darwin [7], in which, from an initial population with randomly individuals, after subsequent generations, new individuals are produced from old ones by means of crossover, selection and mutation operations, based on natural selection, the good individual will have more chances of survival to become part of the next generation. Thus, after successive generations, obtains the best-so-far individual corresponding to the final solution of the problem.

Similar, GP is based on the evolution of a randomly created initial population of individual computer programs composed of the available functions and terminals. Each individual program in the population is executed and assigned fitness according to the problem's fitness measure. With a probability based on fitness one or more individual from the population is selected to participate in the genetic operations like crossover and mutation in order to generate the new population where its individuals' fitness's are evaluated. The GP encoding for the solutions is tree-shaped, so the user must specify the following parameters

1- The set of terminals (leaves of the tree) which consists of the independent variables of the problem, and random constants

2- The set of primitive functions (nodes of the tree) these functions can be ,

3- The fitness measure measuring the fitness of individuals in the population),

4- The replacement strategy which decide a whole new population is generated, or elitism is applied to save the fittest individuals from replacement,

5- The stopping criteria that when is reached the whole evolutionary process stops.

During the evolutionary process of genetic programming individuals increase in tree size and depth without a corresponding increase in fitness [8]. This problem is due to the use of variable-length trees as selective bias towards fitness seems to unavoidably lead the search towards programs with a large size. This phenomenon, called GP bloat. GP bloat slows the evolutionary search process, consumes memory. Also, more complex individuals always suffer from over-fitting over the training data. Furthermore bloat makes these evolved programs even more difficult to comprehend [9]

The most common approach to dealing with bloat in tree-based genetic programming individuals is set an upper bound to the complexity of individuals in the population. An alternative to depth limiting is to introduce an explicit fitness penalty in some way based on excess size [10]. Another approach to control tree bloat is to apply genetic operators designed to target redundant code or the bias against offspring size increases [11].

.In this paper we focus on the parsimony pressure approach [1]. It use a multi-objective method in which two objectives are used, problem-specific error and tree size. Multi-objective GP (MOGP) has a number of advantages: As well as controlling bloat very effectively, it does not require a pre-determined depth-limit parameter and the tree depth is free to adjust to suit the problem at hand.

This paper is organized as follows: We present related work of using genetic programming in feature extraction in section 2. Section 3 describes the proposed methodology to incorporate MOGP with robust classifiers. In section 4 experimental results are shown. Conclusion is given on section 5.

2. RELATED WORK

2.1 GP and Feature Extraction

In this paper we use GP feature extraction uses the classifier output is the Evaluation Criteria: For feature extraction. This approach which is called Wrapper approach has been used with most of the classification algorithms such as decision tree (DT), support vector machines (SVMs), Naïve Bayes (NB), K-nearest neighbor (KNN), artificial neural networks (ANNs), and linear discriminant analysis (LDA), have been applied to wrappers for feature selection [13], [14], [15].

Raymer, Punch, Goodman, and Kuhn [16] applied GP to improve KNN classifier performance by, they evolving a tree for each attribute. While Bot [17] has used GP to evolve new features one-at-a-time to a KNN classifier. In the other hand Kotani, Nakai, and Akazawa [18] used GP to evolve the polynomial combination of raw features to fed into a KNN classifier. Also, Firpi, Goodman, and Echauz [19] developed artificial features applied KNN classifier for predicting epileptic seizures.

Tackett [20] developed a processing tree derived from GP for the classification of features extracted from images. Sherrah [21] applied evolutionary computation to create multi-trees GP each tree encoding one feature. Krawiec [22] constructed a fixed-length decision vector using GP proposing an extended method to protect 'useful' blocks during the evolution.

Recently [23] applied GP feature extraction on a malware detection to improve the accuracy while finding a proper balance between the three basic requirements for malware detection algorithms: the training time on large datasets, the false positive rate, and the detection rate

2.2 GP and Network Intrusion

Song et al. [24] applied page-based linear genetic programming for intrusion detection as a binary classifier between normal actions and attacks. Only the basic features of connections were employed. The half a million records of the 10 % KDD-99 were used and, in each generation, a target number of cases from the dataset were selected randomly for fitness evaluation, preferring cases that were difficult to learn or cases that had not been selected for several generations.

Grosan [25] used multi-expression programming (MEP) to detect intrusions in computer networks. An MEP variable-length chromosome encodes several expressions instead of just the one expression in standard GP. The fitness of the entire individual is supplied by the fitness of the best sub-expression encoded in that chromosome.

Lu and Traore [26] used GP to evolve new rules from initially created rules that cover already-known attacks. New rules are generated by four operators including: Mutation, reproduction, crossover and a dropping condition operator. The fitness of each rule is evaluated by incorporating the support and the confidence factor of the rule in one function.

Faroun [27] employed a genetic programming feature extraction technique to evolve a non-linear mapping from the KDD-99 dataset's 41 features into a one-dimensional decision space where he used a dynamic threshold classifier to separate the five classes in the dataset.

Badran and Rockett [28] used multi-objective genetic programming to evolve a feature extraction stage for multiple-class classifiers by mappings the input space into single decision space. They used Bayesian classifier to incorporate changing priors and/or costs associated with mislabeling without retraining. They applied this approach to the KDD-99 intrusion detection dataset and obtained results which are highly competitive with the KDD-99 Cup winner but with a significantly simpler classification framework.

Thi Anh Le [29], we propose a method by using Genetic Programming for detecting malwares. The experiments were conducted with the malwares collected from an updated malware database on the Internet and the results show that Genetic Programming, compared to some other well-known machine learning methods, can produce the best results on both balanced and imbalanced datasets

Jorge Blasco et al. [30] guided the GP search by means of a fitness function based on recent advances on IDS evaluation. They applied their approach to KDD-99. Results clearly show that an intelligent use of GP achieves systems that are to top state-of-the-art proposals in terms of effectiveness, improving them in efficiency and simplicity.

3. PROPOSED METHODDOLGY

As shown in figure 1, multi-objective genetic programming based feature extraction system applied in addition to the traditional three classifiers implementation. Each individual in GP represents a set of new features, which represents mapping of the original features into decision space to be passed to classifier. These features are calculated through special nodes in the tree denoted by Ai. The number of these nodes are dynamically changing according to the evolution operations. The classification accuracy of the classifier is evaluated and it's considered as one element of fitness vector

for this individual that needs to be minimized. The second element of the fitness vector is the tree complexity that is represented by the node count which needs to be minimized in order to control the GP tree bloat. The huge trees could produce an extremely small error over the training set but a very poor performance estimated over an independent validation set.

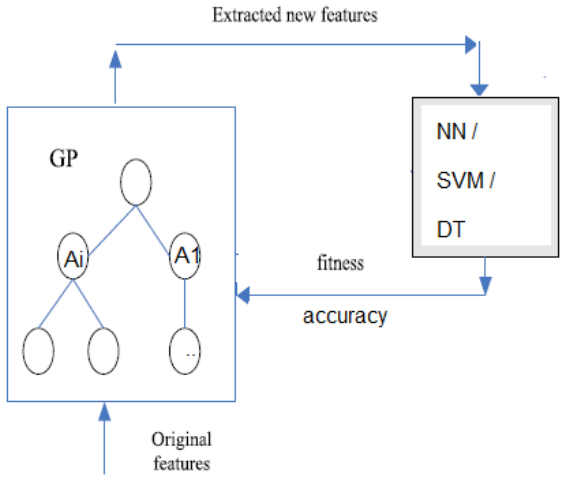

Figure 1 MOGP Incorporated with Robust Classifier

MOGP was implemented in Python with the following parameters in Table 1. While, the other traditional classifiers were used by calling Scikit-learn API 2013 [31] . Scikit-learn integrates a wide range of state-of-the-art machine learning algorithms for medium-scale supervised and unsupervised problems

Table 1. MOGP Implementation Parameters

Parameter	Method / Value
Terminal set	Select randomly from the input features or select random constants [0..1]
Function set	+ , - , / , * , A (feature output)
Initial population	100 individual , with max depth of 5 , with random number of output nodes [1..10]
Replacement Strategy	Used an elitist, steady-state strategy. In each generation, only two individuals are generated and added to the population to replace the worst two individuals
MO Fitness Vector	Classifier Accuracy (max) Tree node count (min)
Selection	Roulette wheel section was employed to select randomly two individuals for crossover the raw fitnesses to decide which should be chosen.
crossover	To select the crossover point in the sub tree We have used a depth-dependent as a sub tree is selected in chosen depth with a higher probability to small trees in the same depth
mutation	Mutation was always applied to the results of the crossover operation in mutation
Stopping	1000 generation

Table 2 shows the used Scikit-learn classes for Neural Networks, Support Vector Machines, and the Decision Trees. While Tables 3-5 show the used parameters for calling these classes APIs. For the three classifiers, the method ***fit*** is used to build the classifier from the training set. While the method ***score r***eturns the mean accuracy on the given test data and labels. These classifiers are first built using the original features in the datasets. Then, they were used during the feature extraction evolution as they return classifier accuracy which is considered as one element of the fitness vector for each MOGP individual

. Table 2. Scikit-learn Classifiers Classes

Classifier	Used scikit-learn class
NN	sklearn.neural_network.MLPClassifier
SVM	sklearn.svm.SVC
DT	sklearn.tree.DecisionTreeClassifier

Table3. Scikit-learn NN API parameters

Parameter	Value
Algorithm	MLP
hidden_layer_sizes	1
activation	‘relu’
batch_size	200
alpha	0.001
learning_rate_init	0.001
momentum	0.9
max_iter	100

Table 4. Scikit-learn SVM parameters

Parameter	Value
Kernal	rbf
Degree	Degree of the polynomial kernel function = 3
Gama	Kernel coefficient = 1/ features
shrinking	True
max_iter	200
tol	1e-3

Table 5. Scikit-learn DT API parameters

Parameter	Value
Algorithm	C4.5
criterion	“entropy” : information gain.
max_depth	7
min_samples_split	2
min_impurity_split	1e-7
min_samples_leaf	1

4. DATA SETS

4.1 UCI Datasets

In order to evaluate the performance of the proposed method on a small datasets, a range of benchmark datasets from the UCI Machine Learning database [32] are used. A variety of datasets was chosen where the number of input features varies from 4 to 21, the number of output classes from 3 to 10, and the number of patterns in the dataset from 150 to 10991. Each dataset was repeatedly split into two folds, one used for training and the other used for testing.

Table 6. UCI datasets

Name	Features	Size	Classes
IRIS	4	150	3
Pen digits (PEN)	16	10991	10
Image segmentation (SEG)	19	2310	7
Thyroid Gland (TGD)	5	215	3
Wine recognition (WIN)	13	179	3
Teaching Assistant Evaluation (TAE)	5	151	3
Thyroid (THY)	21	7200	3
GLASS	9	214	6

4.2 KDD-99 Cup Dataset

Our principal interest is in using multi-objective genetic programming as a feature extractor and selector to perform data mining on this very large intrusion detection dataset []

The KDD-99 benchmark consists of three datasets: The whole dataset that is 5 million records, the 10% that is composed of about half a million records, and the corrected KDD-99. In this paper we are concerned with the 10% KDD-99 as a training dataset and the corrected KDD -99 to test our methodology and to compare with published results on this dataset. KDD dataset covers four major categories of attacks: Probing attacks (information gathering attacks), Denial of-Service (DoS) attacks (deny legitimate requests to a system), user-to-root (U2R) attacks (unauthorized access to local super-user or root), and remote-to-local (R2L) attacks (unauthorized local access from a remote machine). KDD dataset is divided into labeled and unlabeled records. Each labeled record consisted of 41 attributes (features) and one target value. Target value indicated the attack category name.

The KDD-99 dataset exhibits four main challenges for machine learning algorithms as it is a cost-sensitive, multi-objective, a large dataset, and contains mixed types of data.

Table 7. KDD-99 dataset features

Features Categories	Continuous	Discrete	Total
Basic Features	5	4	9
Content features	8	5	13
Time-based features	9	0	9
Host-based features	10	0	10
Total	**32**	**9**	**41**

5. Results

5.1 UCI Datasets

The proposed models (MOGP with robust classifiers) are implemented and worked successfully with the UCI dataset. In table 8, table 9 the comparative of mean misclassification errors between three classifiers over eight UCI datasets is shown. The most result shows that the performance of the three classifiers is enhanced by adding the layer of MOGP feature extraction. NN classifier has best results over six datasets, and SVM classifier give better result in two datasets. These models succeeded in transforming the input features into the decision space where the classifiers performance were enhanced. Considering statistical significance to be two standard errors shows NN classifier has significantly superior to most others classifiers as shown in table 10

Table 8 Mean Error Using Traditional Classifiers Only

dataset	NN	SVM	DT
win	**0**	0.001	0.03
TGD	**0.011**	0.012	0.021
TAE	**0.434**	0.445	0.455
PEN	**0.020**	0.13	0.052
SEG	**0.043**	0.081	0.331
Glass	**0.335**	0.413	0.41
THY	**0.009**	0.01	0.063
IRIS	**0**	0	0

Table 9 Mean Error Using MOGP with Traditional Classifiers

dataset	NN	SVM	DT
win	**0**	0.001	0.002
TGD	**0.009**	**0.009**	0.020
TAE	0.**312**	0.352	0.372
PEN	0.**015**	0.09	0.031
SEG	0.032	**0.051**	0.131
Glass	0.**280**	0.402	0.314
THY	0.009	**0.008**	0.020
IRIS	**0**	**0**	**0**

Table 10 F-statistic comparisons of classifiers on each dataset Error

dataset	NN-SVM	NN-DT	SVM-DT
win	2.362	1.713	4.321
TGD	5.248	7.375	6.872
TAE	7.654	9.461	6.121
PEN	4.698	4.365	4.921
SEG	8.023	6.658	7.32
Glass	6.632	5.624	2.214
THY	1.255	6.152	4.468
IRIS	3.998	1.954	0.478

5.2 KDD-99 Cup Dataset

Generally this MOGP approach with robust classifiers worked successfully with the large dataset. It did not require any pre-processing or sampling; neither did it need any prior knowledge or assume any specific distribution of the classes in the dataset in the input space. It works directly on the data in its original form. The results obtained by applying the traditional classifiers without the MOGP feature extraction phase are shown in table 11.

Table 12 shows the results after applying the MOGP feature extraction phase. It is clear that the performance of the three classifiers were enhanced by adding the MOGP feature extraction. The NN Classifier shows better result in the context as it achieved the Accuracy value 0.93112 and Detection Rate 0.91359. It gives quite near to best result of a mean cost per example which is 0.23618.

The SVM Classifier gives a best enhancement in the results of false positive Rate which is 0.00720 and it better result in of mean cost per example which achieved 0.23252. The confusion matrix of three models has shown in tables 13-15.

Table 11 KDD dataset Results

Classifier / Evaluation	DT	SVM	NN
Accuracy	0.88125	0.89599	0.9021
Detection Rate	0.90010	0.88121	0.90147
False positive Rate	0.02038	0.01115	0.02178
MCPE	0.25101	0.24245	0.24716

Table 12 KDD dataset Results after MOGP feature Extraction

Classifier / Evaluation	DT	SVM	NN
Accuracy	0.92591	0.93999	**0.93112**
Detection Rate	0.90859	0.91081	**0.91359**
False positive Rate	0.01966	**0.00720**	0.00988
MCPE	0.24811	0.**23252**	0.23618

Table 13 Confusion matrix of Decision Tree

	Normal	Prob	DOS	U2R	R2l
Normal	59419	1044	103	22	5
Prob	820	3021	322	3	0
Dos	5692	128	223464	569	0
U2R	93	8	118	5	4
R2l	15361	2	24	2	800

Table 14 Confusion matrix of SVM

	Normal	Prob	DOS	U2R	R2l
Normal	60160	295	58	4	76
Prob	668	3059	164	0	275
Dos	5956	776	223112	0	9
U2R	38	149	2	28	11
R2l	13995	540	1	2	1651

Table 15 Confusion matrix of Neural Networks

	Normal	Prob	DOS	U2R	R2l
Normal	60061	200	310	4	18
Prob	272	3599	292	0	1
Dos	5180	523	224125	0	25
U2R	72	104	5	30	17
R2l	15150	7	20	1	1011

The final results obtained in the experiments conducted in addition to the results of the three winning entries in the KDD-99 contest are shown in table 11. All the experiments give similar results which show the ability of the multi-classification multi-objective genetic programming feature extraction approach to extract the most discriminant features. Also the complexity of the solutions presented is very low compared to those of the first winning KDD-99 entry where 500 decision trees were used, and also to the second winning KDD-99 entry which used 755 decision trees.

Table 16 COMPARISON BETWEEN OBTAINED RESULTS AND WINING ENTRIES RESULTS

Evaluation Method	MCPE	DR	FPR	Accuracy
First place in KDD-99	0.2331	0.918	0.005	0.927
Second place in KDD-99	0.2356	0.915	0.006	0.929
Third place in KDD-99	0.2367	-	-	-
DT	0.248	0.9086	0.0197	0.92591
SVM	0.232	0.9108	0.0072	0.93999
NN	0.236	0.9136	0.0098	0.93112

The NN Classifier give a result in which is very near to the winning entries in the KDD-99 contest especially in Accuracy and Detection Rate but DT classifier has the highest rate in mean cost per example. The SVM has good example which is better than the best winning entry in the contest as it achieved the value 0.232 for MCPE

6. CONCLUSION

We have proposed the incorporation of MOGP that works as feature selection and extraction with Robust Classifiers (NN, SVM and DT). Those models are examined by UCI datasets and KDD dataset. The results show that NN classifier has better result over UCI datasets. In KDD dataset, the three models worked successfully with the large dataset. And it achieved better result in mean cost per example by SVM classifier.

7. REFERENCES

[1] Y Zhang and P I Rockett "Domain-Independent Approaches to Optimise Feature Extraction for MultiClassification using Multi-Objective Genetic Programming" Technical Report No. VIE 2007/001 Department of Electronic and Electrical Engineering University of Sheffield

[2] Y. Liu, F. Tang, and Z. Zeng, "Feature selection based on dependency margin," IEEE Trans. Cybern., vol. 45, no. 6, pp. 1209–1221, Jun. 2015.

[3] H. Liu and Z. Zhao, "Manipulating data and dimension reduction methods: Feature selection," in Encyclopedia of Complexity and Systems Science. Berlin, Germany: Springer, 2009, pp. 5348–5359.

[4] H. Liu, H. Motoda, R. Setiono, and Z. Zhao, "Feature selection: An ever evolving frontier in data mining," in Proc. JMLR Feature Sel. Data Min., vol. 10. Hyderabad, India, 2010, pp. 4–13.

[5] H. Liu and L. Yu, "Toward integrating feature selection algorithms for classification and clustering," IEEE Trans. Knowl. Data Eng., vol. 17, no. 4, pp. 491–502, Apr. 2005.

[6] J. Koza " Genetic programming: On the programming of computers by means of natural selection MIT Press, Cambridge, Massachusetts (1992)

[7] C. Darwin " On the origin of species by means of natural selection or the preservation of favoured races in the struggle for life" Cambridge University Press, Cambridge, UK (1864)

[8] Peter A. Whigham, and Grant Dick, "Implicitly Controlling Bloat in Genetic Programming," IEEE Transaction on Evolutionary Computation, Vol. 14, No. 2, APRIL 2010, pp. 173-190.

[9] M. J. Streeter, "The root causes of code growth in genetic programming," in Proc. Genet. Programming (EuroGP '03), vol. 2610. Essex: SpringerVerlag, Apr. 14–16, 2003, pp. 443–454

[10] H. Stringer and A. Wu, "Bloat is unnatural: An analysis of changes invariable chromosome length absent selection pressure," Univ. Central Florida, Tech. Rep. CS-TR-04-01, 2004.

[11] C. Skinner, P. J. Riddle, and C. Triggs, "Mathematics prevents bloat," in Proc. 2005 IEEE Congr. Evol.Comput., vol. 1. Edinburgh, U.K.: IEEE Press, Sep.2–5, 2005, pp. 390–395

[12] Elsayed S, Sarker R, Essam D (2015) Survey of uses of evolutionary computation algorithms and swarm intelligence for network intrusion detection. International Journal of Computational Intelligence and Applications 14(04):1550,025, D

[13] H. Liu and Z. Zhao, "Manipulating data and dimension reduction methods: Feature selection," in Encyclopedia of Complexity and Systems Science. Berlin, Germany: Springer, 2009, pp. 5348–5359.

[14] H. Liu, H. Motoda, R. Setiono, and Z. Zhao, "Feature selection: An ever evolving frontier in data mining," in Proc. JMLR Feature Sel. DataMin., vol. 10. Hyderabad, India, 2010, pp. 4–13.

[15] B. Xue, M. Zhang, and W. N. Browne, "Particle swarm optimization for feature selection in classification: A multi-objective approach," IEEETrans. Cybern., vol. 43, no. 6, pp. 1656–1671, Dec. 2013.

[16] Raymer, M., Punch, W., Goodman, E., & Kuhn, L.. Genetic programming for improved data mining: Application to the biochemistry of protein interactions. In Proceedings of the first annual conference on genetic programming (pp. 375–380). Cambridge, Massachusetts: MIT Press. (1996)

[17] Bot, M., & Langdon, W. Application of genetic programming to induction of linear classification trees. In Genetic programming, proceedings of EuroGP'2000 (pp. 247–258). Berlin, Heidelberg: Springer-Verlag. Chui, C. (1992). An introduction to wavelets. Boston: Academic Press.

[18] Kotani, M., Nakai, M., & Akazawa, K. . Feature extraction using evolutionary computation. In Proceedings of the 1999 congress on evolutionary computation, 1999. CEC 99 (Vol. 2, pp. 1230–1236).

[19] Raymer, M., Punch, W., Goodman, E., & Kuhn, L. Genetic programming for improved data mining: Application to the biochemistry of protein interactions. In Proceedings of the first annual conference on genetic programming (pp. 375–380). Cambridge, Massachusetts: MIT Press.1996

[20] Tackett, W. Genetic programming for feature discovery and image discrimination. In Proceedings of the fifth international conference on genetic algorithms, ICGA-93 (pp. 303–309). 1993

[21] Sherrah, J. Automatic feature extraction for pattern recognition. Ph.D. Thesis, The University of Adelaide.1998

[22] Krawiec, K. Genetic programming-based construction of features for machine learning and knowledge discovery tasks. Genetic Programming and Evolvable Machines, 3(4), 329–343.2002

[23] Cristina Vatamanu, Dragos Gavrilut, Razvan Benchea, Henri Luchian, "Feature Extraction Using Genetic Programming with Applications in Malware Detection", , vol. 00, no. , pp. 224-231, 2015.

[24] D. Song, M. Heywood and A. N. Zincir-Heywood, A linear genetic programming approach to intrusion detection, Genetic and Evolutionary Computation — GECCO 2003 (2003) 2325–2336.

[25] A. Abraham, C. Grosan and C. Martin-Vide, Evolutionary design of intrusion detection programs, Int. J. Netw. Security 4 (2007) 328–339.

[26] W. Lu and I. Traore, Detecting new forms of network intrusion using genetic programming, Comput. Intell. 20 (2004) 475–494.

[27] A. Boukelif and K. M. Faraoun, Genetic programming approach for multi-category pattern classification applied to network intrusions detection, Int. J. Comput. Intell. Appl. 6 (2006) 77–99

[28] K. Badran, P. Rockett, "Multi-class pattern classification using single multi-dimensional feature-space feature extraction evolved by multi-objective genetic programming and its application to network intrusion detection", Genetic Programming and Evolvable Machines, vol. 13, no. 1, pp. 33-63, 2012.

[29] Thi Anh Le, Thi Huong Chu, Quang Uy Nguyen, Xuan Hoai Nguyen, "Malware detection using genetic programming", *Computational Intelligence for Security and Defense Applications (CISDA) 2014 Seventh IEEE Symposium on*, pp. 1-6, 2014.

[30] Jorge Blasco, Agustin Orfila, Arturo Ribagorda "Improving Network Intrusion Detection by Means of Domain-Aware Genetic Programming" DOI 10.1109/ARES.2010.53 in IEEE 2010.

[31] Buitinck *et al.*, "API design for machine learning software: experiences from the scikit-learn project", 2013.

[32] KDD data set, 1999; http://kdd.ics.uci.edu/databases/-kddcup99/kddcup99.html