Rule Extraction from Neural Networks and Other Classifiers Applied to XSS Detection

Fawaz A. Mereani[1,2] [0000−0003−2832−304X] and Jacob M. Howe[1] [0000−0001−8013−6941]

[1] City, University of London, London, United Kingdom
[2] Umm AL-Qura University, Makkah, Saudi Arabia
{fawaz.mereani, j.m.howe}@city.ac.uk

Abstract. Explainable artificial intelligence (XAI) is concerned with creating artificial intelligence that is intelligible and interpretable by humans. Many AI techniques build classifiers, some of which result in intelligible models, some of which don't. Rule extraction from classifiers treated as black boxes is an important topic in XAI, that aims to find rule sets that describe classifiers and that are understandable to humans. Neural networks provide one type of classifier where it is difficult to explain why the inputs map to the decision; support vector machines provide a second example of this kind. A third type of classifier, k-nearest neighbour (k-NN), gives more interpretable classifiers, but suffers from performance problems as the model is little more than a representation of the training data. This work investigates a technique to extract rules from classifiers where the underlying problem's feature space is Boolean, without looking at the inner structure of the classifier. For such a classifier with a small feature space, a Boolean function describing it can be directly calculated, whilst for a classifier with a larger feature space, a sampling method is investigated to produce rule-based approximations to the behaviour of the underlying classifier, with varying granularity, leading to XAI. The behaviour of the technique with neural network, support vector machine, and k-NN classifiers is experimentally assessed on a dataset of cross-site scripting (XSS) attacks, and proves to give very high accuracy and precision, often comparable to the classifier being approximated.

Keywords: Rule Extraction · Explainable AI · Neural Networks · XSS

1 Introduction

Explainable Artificial Intelligence (XAI) is the term used to capture the problem of making artificial intelligence applications intelligible to humans [15]. XAI aims to "produce more explainable models, while maintaining a high level of learning performance (prediction accuracy); and enable human users to understand, appropriately trust, and effectively manage the emerging generation of artificially intelligent partners" [15].

Machine learning, especially neural networks, can produce classifiers that give high predictive accuracy, leading to excellent performance in complex tasks such as detecting objects in the images [45, 17], or understanding natural language [9]. The resulting trained models (again, particularly from neural networks) are often essentially black boxes. The way in which a neural network reaches a decision from the input data is not

accompanied by an explanation that can be interpreted by a user. This also applies to an extent to other machine learning techniques such as support vector machines. There is a lot of interest in being able to give an explanation for the decision making resulting from machine learning models. That might be by opening up black box models [4, 3], by developing methods that help to understand what the model has learned [25, 34], or as is done in the current paper, by extracting rules from the networks.

In [29] the authors introduced a technique for rule extraction from neural networks (NN) in the case that the feature set is Boolean. This initial investigation begins with the observation that if the features that form the input to a neural network are all Boolean, and the output value is a Boolean, then the trained neural network precisely defines a Boolean function. It is demonstrated that if the number of features is small, then each possible input combination can be evaluated, essentially enumerating the truth table for the Boolean function represented by the neural network. As the number of features increases, the size of the truth table rises rapidly, hence its enumeration becomes infeasible. Therefore, for neural networks defined over larger feature spaces approximations of the encoded Boolean function were considered which use a sampling based approach. The extracted rules are Boolean functions and, with each Boolean variable corresponding to a feature in the problem domain, these rules are clearly interpretable.

The target application is the detection JavaScript based cross-site scripting (XSS) attacks. In previous work, a variety of machine learning techniques were applied to determine whether a script was malicious or benign [30, 31]. The performance of the resulting classifiers was evaluated and they achieved high predictive accuracy results using a large real-world dataset of scripts. It was found in this work that the most successful set of features to abstract the scripts to was Boolean valued (sixty-two features in total), hence the models are Boolean valued.

The current paper develops this works further. The approximation of classifiers which cannot be exactly described is based on a sampling of the values given by the classifier. This sampling is further explored, in terms of the size of the sample needed in order to determine the classification value, how accurate this classification is, and how the sampling determines the classification value.

In [29] it was conjectured that the technique would work with any classifier with Boolean features. In earlier work, support vector machines (SVM) and k-nearest neighbour (k-NN) classifiers have proved to be particularly good at classifying XSS [30, 31]. SVM share, to some extent, the problems of interpretability that neural networks have. Whilst k-NN classifiers are fairly easy to interpret (so there is not a big XAI challenge in using them), unlike many other classifiers they grow with the size of the training data, since they are essentially a representation of that data. This leads to performance problems, and motivates rule extraction for a different reason, namely that an extracted rule would not be of the same size as the training data and should perform classification more efficiently.

The aim of this paper is to perform rule extraction from classifiers treated as black boxes, with the extracted Boolean functions giving a decision making method that is more explainable to humans [15]. The contributions of this paper are as follows:

- Validation of the observation that for an entirely Boolean feature set, with binary classification, the trained classifier defines a Boolean function

- An investigation into how to use a sampling approach to approximate this Boolean function when the feature set is sufficiently large for the generation of the precise function to be infeasible
- A demonstration that the Boolean function extraction can be performed for neural network, SVM, and k-NN classifiers for XSS detection problems
- An empirical evaluation of approximate rule extraction from the XSS classifiers.

The rest of this paper is organised as follows: Section 2 gives background and related work on methods for extracting rules and the detection of XSS attacks in scripts. Section 3 describes the dataset used, including how features are selected and ranked, how classifiers are trained and evaluated using this dataset, and the method used for constructing and approximating Boolean functions. Section 4 presents results related to the application of the rule extraction, and Section 5 discusses the results. Further discussion and concluding remarks are given in Section 6.

2 Background and Related Work

It has been said that requirements of a software system (accuracy, and ease-of-use) almost always work in a contradictory manner. As [7] has stated, "Unfortunately, in prediction, accuracy and simplicity (interpretability) are in conflict." The extraction of rules from a trained classifier can be an intermediate method which allows for the satisfaction of both of these requirements via the use of a relatively simple and understandable set of such rules which simulates a model's predictions (that is, they explain the rules which are used inside a black box) [11, 27, 5].

The neural network classifier model represents one of the most popular types of classifier from which rules can be extracted. Algorithms for extracting rules from neural network classifiers may be divided into three main categories:

1. **Pedagogical:** This kind of method is not concerned with the internal structure of the network, but only with deriving the rules used by the network by looking at the relationships between the inputs and the outputs. It does not scrutinise the internal behaviour of the network [46, 48]. An example of the use of this type of rule extraction can be found in [40], where rules were extracted from a multilayer medical diagnostic system by monitoring the impact on network outputs of changes to its inputs. The VIA technique [47] is another example of a pedagogical method, where the generation and testing of an input dataset was focused on the extraction of rules from the neural network while it was being trained using backpropagation. Other techniques in this category are sampling and the reverse engineering of neural networks [16]. An example of the use of samples for a pedagogical approach is given in [10], where Craven and Shavlik proposed an algorithm called TREPAN. This algorithm extracts M-of-N and split trees from an ANN with one hidden layer, which the network employed as an "oracle" to statistically validate the correctness and significance of the generated rules. Saad and Wunsch proposed, in [39], a method they termed HYPINV which relied on a network inversion technique. This method is capable of extracting the hyperplane rule learned by a multilayer perceptron in the form of conjunctions and disjunctions of hyperplanes.

The present study will focus on the use of samples to extract rules from black box classifiers. Hence, knowing how to use samples to extract rules is essential. Huysmans et. al. [12] specifies methods for using samples to extract rules; additional training instances are created to act as samples for use by TREPAN. Another method is to create random instances; this method keeps the samples nearer to the original training instances, which in turn ensures that the generated samples are similar to the original training data when used with ANN-DT in [41].

2. **Decompositional:** The methods in this category are concerned with extracting the rules directly from the layers within the network. Such decompositional methods analyse network activation, the outputs of hidden layers and the associated weights [13]. An example of this type of method is found in [43, 44] where a three step algorithm was proposed for analysing and thus 'understanding' the neural network. Deriving rules from deep neural networks is one of the more difficult tasks in this area. Katz et. al proposed a new algorithm in [22] to verify the properties of neural networks using ReLUs activation functions [33] by the application of simplex. The algorithm was modified to support ReLU constraints and termed Reluplex. Reluplex is concerned with reducing the search space, but it needs to 'split' using a specific ReLU constraint. Note that in this method many or indeed all of the ReLUs involved can be ignored. This algorithm has been applied to the next generation of airborne collision avoidance systems for unmanned aircraft [20].
3. **Eclectic:** Methods of this type combine attributes derived from the two previous types. This type of rule extraction was used in [23], where a method for discovering trends within a large dataset was proposed which employed a neural network as a black box which had the function of discovering knowledge. At the same time, the method examines weights by pruning and clustering the activation values of the hidden units within the network.

It is fruitful to compare the types of extraction methods used in terms of their relative advantages and disadvantages. First, it can be observed that the extraction of rules using decompositional approaches is complex and requires considerable computational resources, and this is the most important constraint with regard to the use of these methods. Pedagogical approaches are generally faster because they do not attempt to analyse the weights and internal structure of the associated neural network. However, the most important disadvantage of this approach is that it is less likely to find all the rules that describe the behaviour of the network correctly. The eclectic approach is slower but more precise because it combines the two other methodologies [2].

A decision tree is one of the most common methods of representing the rules extracted from non-rule-based classifiers, where the individual rules can be specified in the form $if...then$. The decision tree itself is built using these rules such that the classes (returned by the classifier) are the leaves and the branches represent the sequences of features (conditions) that lead to these classes [1]. Representing the rules in a way which is understandable by human-being is described in [6] and [18].

1. **If-Then / If-Then-Else:** Rules are represented by using "if" condition. The condition component is a set of conditions on input variables, followed by a "$then$" which indicates a class. An example of an "$if...then...else$" rule is:

$if(a_{11} < x_1 < a_{12})$ *and* $(a_{21} < x_2 < a_{22})$ *then Class A else Class B.*
Note that most extraction algorithms create rules that contain conjunctions and they will generally ensure that the conditional parts define separate areas in the input space, meaning that the rules are mutually exclusive. Therefore, only one rule will be able to classify a new entry.

2. **M-of-N:** This type of represention of the rules is considered to be more compact than "$if...then$" rule sets; the decision in relation to just one class is made such that it is required that M of the full set of N rules be true for this class to be returned.
3. **Oblique rules / multi-surface:** This type of representation is made using rules that separate a feature space using planes, each side of each plane represents a particular class, this allows each data point in the space to belong to a specific class. This representation is more difficult to understand, but such rules are powerful.
4. **Equation rules:** This type of rule representation is similar to oblique rules, but non-linear equations in the condition part are used for this purpose. This type of rule representation makes it difficult to understand the extracted rules, thus contributes little to the interpretation of the original model.
5. **Fuzzy rules:** This method of representing rules is similar to that of ($if...then$) rules, the difference being that this representation deals with fuzzy sets and its underlying mechanism is many valued fuzzy logic. Fuzzy rules are easy to understand because they are expressed using concepts that are readily comprehensible by the user.

In this study, the pedagogical approach combined with a sampling technique will be adopted to extract the rules from a neural network classifier. The proposed approach will focus on extracting the rules by finding the relationships between the inputs and the returned classes. The rules so extracted will be represented in the form ($if...then..else$), since Boolean functions act as decision trees.

2.1 Overview of Minimising Boolean Expressions

To better understand the Boolean function being used, it is useful to extract the rules into a compact representation. A minimal representation of a Boolean expression is easier to understand and to write out; in addition, explanations based on such minimal forms are less prone to error. Importantly, a minimal representation can be more effective and efficient when implemented in experiments [38]. Therefore, the minimisation of Boolean expressions, to find a representation equivalent to the original expression but of a minimum size, is considered here.

Minimisation can be achieved in several ways, where the important factor in relation to choosing a method is the number of variables in the expression. The commonly used methods for minimising Boolean expressions are:

1. **Karnaugh Maps:** This is a graphical method for minimising Boolean expressions [21], whereby the truth table of the expression is expressed as a matrix, all the complementary pairs are then eliminated, and the result is a minimised Boolean expression. This method is very effective when only small numbers of variables are involved, but it becomes more unwieldy when there are large numbers of variables.

2. **Tabular (Quine-McCluskey):** This method is, in general, more effective than the Karnaugh maps method, and in particular its effectiveness can be observed when minimising expressions containing a large number of variables [26]. Minimising Boolean expressions using this method is achieved via two main activities: the identification of primary implicants and the selection of essential primary implicants. Essential primary implicants are all those terms that will be present in the final simplified function. The starting point is to list the minterms that define the function, then the prime implicants are found by a matching method. Each minterm and maxterm are compared with every other minterm. When the expressions differ in terms of only one variable, this variable will be removed and a function will be created which excludes it. This process is repeated for each minterm and maxterm pair until the search ends. The selection of essential primary implicants is achieved by creating a table containing the prime implicants. The prime implicant can then be reduced by removing the essential prime implicants, removing the rows that dominate others, and removing the columns that dominate others. These steps are repeated until there no further reduction possible. The weakness of this method is that the run time grows exponentially with the number of variables [19]
3. **Reduced Ordered Binary Decision Diagrams (ROBDDs):** This method is undertaken by imposing an order on the variables of a Boolean function, and then representing this function as a graph structure; this provides a canonical non-redundant representation of the Boolean function, given the variable ordering [8].

In this study, the tabular method will be adopted, because of its effectiveness when minimising expressions with large numbers of variables.

2.2 Cross-Site Scripting

Cross-Site Scripting (XSS) is a type of attack targeting web applications, ranked by OWASP as one of the top 10 attacks [35]. XSS is standardly prevented from being executed through good coding practice, using sanitization and escaping to prevent untrusted content being interpreted as code [51]. Parser-level isolation provides an alternative, confining user input data during the lifetime of the application [32]. Blacklists are viewed as easy to circumvent and these approaches are preferred [51].

Machine learning techniques have been applied to prevent XSS attacks. An early approach [24] evaluates ADTree, SVM, Naive Bayes, and RIPPER classifiers by tracking the symbols that appear in malicious and benign scripts, and achieved precision of up to 92%. Another approach, [50], extracts features used in malicious scripts much more than benign, such as the DOM-modifying functions and the eval function; this method achieved accuracy rate of up to 94.38%. Furthermore, in [30] a number classifiers were evaluated: SVM with linear and polynomial kernels, k-NN and Random Forest. Using a k-NN classifier achieved high accuracy results up to 99.75%, with precision rate up to 99.88%. Here the extracted features depend on the occurrence or not of a syntactic element within a script. A neural network classifier was evaluated in [31] to prevent XSS attacks by using ensemble and cascading techniques and the results gave a very high accuracy of up to 99.80% in the base level which their feature groups used directly, and 99.89% at the meta level where the features are the outputs of base level.

As well as in scripting, there is emerging interest in using neural networks to detect malware in executables, for instance, in [36] a recurrent neural network is used to detect malicious executables at execution time with 93% accuracy.

3 Methodology

This section describes the dataset used in the experiments, the approach to selecting features to build analyses with, the training of the classifiers, and the way in which they are evaluated. The aim of this work is to find Boolean functions as rules extracted from classifiers, which can then themselves be used as classifiers, replacing the originals. The approach to extracting a Boolean function from a classifier (neural network, support vector machine or k-nearest neighbour) is given, both for exact rule extraction, and for a series of approximations to classifiers.

3.1 Datasets

The current work uses the dataset from [29]. This is primarily the dataset from [31], with the training set augmented with files from CSIC 2010 [14] (consisting of 152 malicious instances and 3971 benign instances) in order to to cover more types of scripts to extract more precise rules. The classifiers being trained are to determine whether or not text entered into a web application represents a cross-site script. Hence the dataset consists of 43,218 files, of which 28,068 are labelled as benign and 15,150 are labelled as malicious. Note that 9,068 of the benign scripts are plain text from [49]. These are then divided into a training set of 19,122 instances (5,150 malicious and 13,972 benign) and a holdout testing set of 24,096 instances (10,000 malicious and 14,096 benign). There is no overlap between the training and testing datasets.

3.2 Selected Features

As in [29], the starting point of this work is to abstract the input script file into the 62 features used in [31]. These are divided into two groups: alphanumeric, a range of keywords and tokens from the target application, and non-alphanumeric features, the full set of non-alphanumeric characters. Rather than working with these features immediately without further reflection as in [31], here the features have been ranked by using Algorithm 1 [28]. The method selects the most powerful features in a sequential feature selection. This method works by minimising over all feature subsets, which uses the deviance and chi-square to find the most powerful features. The deviance is twice the difference between the log likelihood of that model and the saturated model, and the inverse of the chi-square with degrees of freedom is used to set the termination tolerance parameter. The application of the ranking algorithm on the feature set shows that only 34 features need be used, and the ranking of these selected features in order of effectiveness is given in Table 1. The key observation of these features is that they are all Boolean valued, that is, the feature occurs in the script or it does not, allowing the exploitation of this additional 0/1 valued structure.

Algorithm 1: Ranking Features Algorithm

Input: Original features set;
Start with empty features subset;
Feature = Sequential Feature Selection;
while *(Deviance > Chi-Square)* **do**
    Feature Subset = Add feature to selected feature subset;
    Feature = Sequential Feature Selection;
end

Table 1. Selected Features [29]

No.	Features	No.	Features
1	Alert	18	%
2	<	19	(&lt)
3	{	20	@
4	?	21	Onload
5	!	22	StringfromCharCode
6	JS File	23	:
7	HTTP	24	\
8	-	25	]
9	’	26	(
10	;	27	‘
11	&	28	Img
12	,	29	′ >
13	Src	30	==
14	Space	31	/
15	&#	32	Onerror
16	Eval	33	//
17	.	34	iframe

3.3 Training Classifiers

Using the features from Table 1, a series of classifiers were trained for each of: feed forward neural networks, support vector machines and k-nearest neighbour. The neural network classifiers were built using a single hidden layer, with 10 hidden units, and the train function (which updates the weight and bias values) was optimised by setting it to be “trainbr” to minimise a combination of squared errors and weights. The support vector machines were trained using a linear kernel. The k-nearest neighbour classifier was training with k optimised to be 4.

For each of the three machine learning techniques, two classifiers were built: one using all 34 features, which is viewed as the best classifier, the one from which rules are to be extracted, and the other using the top 16 features, which will be used for comparison, evaluation and discussion.

3.4 Classifiers and Boolean Functions

Observe that a classifier each of whose input features is Boolean, and whose output is a Boolean value, is precisely equivalent to a Boolean function. Enumerating each possible input, and calculating the corresponding output results in the truth table for this Boolean function. Hence, the classifier can be replaced by this Boolean function. The result is a rule based system, each of whose decisions is explainable and auditable. This is particularly attractive when the initial classifier is a neural network where individual decisions are not explainable. The approach is also attractive when the initial classifier is a SVM, since the Boolean classifier gives more easily understood decisions. When k-NN provides the initial classifier, this classifier is already explainable, but k-NN classifiers suffer in performance, since they grow linearly with the size of the training dataset; the Boolean classifier is of a fixed sized, hence may perform better when there is considerable training data.

In the current study, the feature set is Boolean, therefore this approach applies. Whilst for a small number of features this rule extraction technique might be applied directly, the number of potential inputs grows exponentially with the number of features, and the problem quickly becomes infeasible. This motivates a sampling based approach.

3.5 Sampling

The key classifier in this work is the neural network trained over a feature space with 34 features. This provides an exemplar case for a rule extraction where the Boolean function defined is too large to generate from the classifier. Despite this, there is motivation to find a Boolean function that can be used in place of the neural network. With varying motivation, as discussed in the previous section, this also applies to the SVM and k-NN classifiers trained over 34 features. The approach taken is to sample the classifier and to use this sample to build a Boolean function; this Boolean function then provides an approximation of the original function. The idea is to fix a number of features for which producing a Boolean function is feasible (via a truth table in this case) and to determine what value the function should take by interrogating the initial classifier with the full feature set. For example, suppose it is determined that considering 4 features will result in a truth table that can be feasibly constructed. Then the four highest ranking features (in Table 1) will provide the entries for the truth table. For a row of the truth table, the values of these features is fixed, and then extended with values for the remaining 30 features to give an input to the classifier, which is then queried and the result noted. This is done repeatedly and from the resulting sample the most frequently occurring result is the entry in the truth table.

Whilst the training dataset is relatively large, with 19,112 scripts, this is still sparse compared to the 2^{34} possible inputs to the classifiers considered. This means that whilst the classifiers learn from the training set, the generalisation is not necessarily great enough that every input to the classifiers is equally meaningful. That is, a random sampling extending the fixed values might not give good results, since it might not match the shape of likely inputs. Indeed, this was observed in development of approximations of neural networks in [29], with inputs holding the default value dominating. In order

to counteract this, the extensions were generated from the training set, with a random selection of instances from the training set being selected (with the full 34 features), and these being used for sampling the classifiers with the fixed features replacing the corresponding feature values.

Algorithm 2 specifies the sampling method. Here, the input to the algorithm is L (an integer) the number of fixed features, Classifier which is a trained classifier (in this case either neural network, SVM or k-NN trained over 34 features) and Sample which is a random selection from the training set of inputs to the neural network. A truth table, *TT*, for the fixed features, with undefined output values, is constructed by buildInitTruthTable. Each *row* of this truth table is considered in turn. The values of the *row* of *TT* are substituted into each element of Sample leading to an *input* which is passed to Classifier for classification. If the *result* is classification as malicious then a counter for malicious instances, *malicious_count*, is incremented, otherwise, *benign_count* is incremented. Once each element of Sample has been considered, a comparison between the two counts is made, and the output column of the truth table *TT* is populated with 0 if most instances are malicious, and 1 otherwise.

In [29], Sample in Algorithm 2 consisted of 1024 inputs chosen at random from the training dataset. As well as repeating this experiment for neural networks, a tactic where a much smaller sample is used investigated for all classifier considered. The 1024 inputs are themselves randomly sampled, with just 32 samples chosen. If this gives a clear cut answer (in these experiments, if 25 or more of the samples give the same output) then this answer is used, if not the sample size is doubled to 64, and if this fails to give a clear cut answer, the sample is again doubled to 128. Using this tactic, this work investigates successive approximations, with a varying number of fixed features: 1, 2, 4, 8, 10, 12 and 16 features.

Fixing the size of Sample to be 32, this work will also investigate how precisely the sampling works by two measures. Firstly, by tabulating for each row of the truth table, how many samples gave malicious as the output. Secondly, by charting for each test case the split between samples giving malicious, and samples giving benign, for the rule that gave the output for the test case.

3.6 Extracting Rules

After labelling all rows in the truth table, each row can be considered to be a rule that describes one class. To give a more succinct set of rules, the Boolean function can be minimised [42] resulting in simplified expressions. The minimised Boolean functions are then evaluated as classifiers. For minimising Boolean functions “Logic Friday” [37] has been used which applies the Tabular Method as a minimisation algorithm.

4 Results

In the experiments, MatLab 2018b was used to build the classifiers, and to find the truth tables based on these classifiers. This was done using various numbers of fixed features: 1, 2, 4, 8, 10, 12, and 16. The extracted truth tables define sets of rules acting

Algorithm 2: Sampling Method Algorithm

```
Input: L ∈ ℕ, Classifier, Sample;
TT = buildInitTruthTable(L);
for row in TT do
    malicious_count = 0;
    benign_count = 0;
    for s in Sample do
        input = substitute(row, s);
        result = Classifier(input) ;
        if result == malicious then
            malicious_count ++;
        else
            benign_count ++;
        end
    end
    if malicious_count > benign_count then
        TT[row] = 0 ; \\Malicious
    else
        TT[row] = 1 ; \\Benign
    end
end
```

as classifiers approximating the original classifier, and these rule sets were then reduced to a more compact representation using "Logic Friday" [37].

Results on trained classifiers and rule extraction are presented in turn for neural networks, support vector machines, and k-nearest neighbour. This is followed by results on how long it takes to perform the rule extraction, and on how clear cut the sampling method for rule extraction is.

4.1 Neural Networks

For neural networks, the first results repeat the experiments from [29] on training neural network classifiers and extracting rules, then a revised set of experiments on rule extraction is given.

4.1.1 Training Neural Network Classifiers As in [29], a neural network classifier was trained using the full 34 features, and tested using the testing dataset. Table 2 gives the performance of this classifier. Evaluation uses the confusion matrix, leading to Accuracy, Precision, Sensitivity, and Specificity measures. This network is the one from which rules are extracted, leading to a series of approximations to it.

In addition, for later comparison, a neural network classifier was trained using just the 16 highest ranked features. Table 3 gives the performance of this classifier. For this network, the Boolean function that the network defines can be precisely extracted. This results in a Boolean classifier whose performance will be the same as the neural network. Table 4 shows the number of the rules that result from constructing the truth

Table 2. Neural Network Classifier Performance Using 34 Features [29]

Accuracy	Precision	Sensitivity	Specificity	Confusion Matrix		
					Malicious	Benign
99.88	99.98	99.75	99.98	Malicious	9998	2
				Benign	25	14071

table for the 16 features, along with the number of rules that classify scripts as benign after minimisation is applied (hence any script whose features do not match a rule for benign is malicious).

Table 3. Neural Network Classifier Performance Using 16 Features [29]

Accuracy	Precision	Sensitivity	Specificity	Confusion Matrix		
					Malicious	Benign
99.78	99.94	99.53	99.95	Malicious	9994	6
				Benign	47	14049

Table 4. Classifier Rules Using 16 Features [29]

Class	Malicious	Benign	Minimised
Rules	41,549	23,987	2,560

4.1.2 Initial Rule Extraction Rules were extracted from the neural network trained on 34 features by applying the sampling method for each row in the truth table, hence the number of extracted rules is equal to 2^n, where n is the number of fixed features in the approximation, and each row describes one rule. As in [29], 1024 samples were used for each row of the truth table. This process was repeated where the number of fixed features was 1, 2, 4, 8, 10, 12, and 16 features. Each of these gives an approximation to the neural network, and the purpose of this repetition is to observe the number of rules that are extracted and the accuracy of the results on the testing dataset.

Table 5 gives the results of testing the rules extracted from the 34 feature neural network, approximating with 1, 2, 4, 8, 10, 12 and 16 features. Again, the evaluation is given in terms of the confusion matrix, and the Accuracy, Precision, Sensitivity and Specificity measures.

Table 5. Results for Rules Extracted from NN using 1, 2, 4, 8, 10, 12, and 16 Features [29]

	Accuracy	Precision	Sensitivity	Specificity	Confusion Matrix		
1 Feature	91.96	80.70	99.92	87.95		Malicious	Benign
					Malicious	8070	1930
					Benign	6	14090
2 Features	91.96	80.70	99.92	87.95		Malicious	Benign
					Malicious	8070	1930
					Benign	6	14090
4 Features	98.95	97.54	99.92	98.28		Malicious	Benign
					Malicious	9754	246
					Benign	7	14089
8 Features	98.13	95.62	99.87	96.98		Malicious	Benign
					Malicious	9562	438
					Benign	12	14084
10 Features	99.15	98.00	99.96	98.60		Malicious	Benign
					Malicious	9800	200
					Benign	3	14093
12 Features	99.82	99.62	99.96	99.73		Malicious	Benign
					Malicious	9962	38
					Benign	3	14093
16 Features	99.90	99.94	99.82	99.95		Malicious	Benign
					Malicious	9994	6
					Benign	18	14078

Table 6 summarises the number of rules for each class by using the various numbers of selected features. The final column gives the number rules that classify the input as benign after minimisation (hence, any input not matching one of these rules is classified as malicious).

Table 6. Numbers of Rules as Related to Numbers of Selected Features [29]

Features	Malicious	Benign	Minimised
1 Feature	1	1	1
2 Features	2	2	1
4 Features	7	9	3
8 Features	100	156	29
10 Features	384	640	62
12 Features	1,560	2,536	229
16 Features	39,792	25,744	2,488

Also included is Table 7 detailing the time take to perform the rule extraction from the 34 feature neural network using 1024 samples for each row of the truth table. The

point to note here is that the extraction of the 16 feature Boolean approximation of the neural network takes considerable time, more than five days of computation.

Table 7. Timing of Rule Extraction from the NN Classifier with 1024 samples [29]

Features	Interval
1 Feature	18 sec
2 Features	37 sec
4 Features	120 sec
8 Features	390 sec
10 Features	7,846 sec
12 Features	30,598 sec
16 Features	482,618 sec

4.1.3 Revised Rule Extraction The time taken to extract Boolean rules with the 16 features using 1024 samples for each row of the truth table motivates investigation of using a smaller sample size. Initial experimentation (not included, but used to generate Figure 1) reduced the sample size to 32. As noted in section 4.5, some decisions based on this reduced number of samples are close, so a tactic that expands the number of samples to 64 if fewer that 25 samples indicate one value or the other is used, and again if fewer that 50 samples indicate one value or the other, the number of samples is expanded to 128.

Table 8 shows the result of applying the Boolean classifiers extracted from the 34 feature neural network to the testing dataset, where the experiment was conducted using 1, 2, 4, 8, 10, 12, and 16 features. Note that the results are close to those of the neural network classifier which used 1024 instances as samples for decision making.

Table 9 shows the number of Boolean rules extracted for each class, and number of rules after minimized.

4.2 Support Vector Machines

This section repeats the experiments performed in the previous section with neural networks, but instead using a 34 feature SVM as the base classifier.

4.2.1 Training Support Vector Machine Classifiers As for neural networks, two SVM classifiers are trained, the first using 34 features, which will be used as the base classifier for rule extraction, and the second using 16 features for comparison purposes. The performance of the SVM trained on the full 34 features is given in Table 10.

Table 11 gives the performance of the SVM trained over 16 features. Again, the Boolean function that this classifier defines can be extracted precisely, and the number of rules that this gives can be seen in Table 12.

Table 8. Extracted Results From NN Using 32/64/128 Samples

	Accuracy	Precision	Sensitivity	Specificity	Confusion Matrix		
1 Feature	58.49	0	0	58.49		Malicious	Benign
					Malicious	0	10000
					Benign	0	14096
2 Features	98.95	97.60	99.87	98.32		Malicious	Benign
					Malicious	9760	240
					Benign	12	14084
4 Features	98.74	97.08	99.88	97.96		Malicious	Benign
					Malicious	9708	292
					Benign	11	14085
8 Features	96.61	99.61	92.76	99.70		Malicious	Benign
					Malicious	9961	39
					Benign	777	13319
10 Features	98.37	96.60	99.46	97.63		Malicious	Benign
					Malicious	9660	340
					Benign	52	14044
12 Features	99.84	99.75	99.87	99.82		Malicious	Benign
					Malicious	9975	25
					Benign	13	14083
16 Features	99.87	99.86	99.83	99.90		Malicious	Benign
					Malicious	9986	14
					Benign	17	14079

Table 9. Numbers of NN Rules Using 32/64/128 Samples

Features	Malicious	Benign	Minimised
1 Feature	0	2	1
2 Features	3	1	1
4 Features	8	8	4
8 Features	103	153	29
10 Features	382	642	84
12 Features	1,576	2,520	245
16 Features	39,861	25,675	2,766

4.2.2 Rule Extraction Boolean rule extraction is performed as for neural networks. The 34 feature SVM classifier is sampled using the 32/64/128 sample tactic and the results are presented in Table 13. The number of rules extracted are given in Table 14.

4.3 k-NN

Again, the same set of experiments were conducted for k-NN. Classifiers are built, then rule extraction performed, with the resulting classifiers evaluated.

Table 10. SVM Classifier Performance Using 34 Features

Accuracy	Precision	Sensitivity	Specificity	Confusion Matrix		
					Malicious	Benign
99.90	99.90	99.88	99.92	Malicious	9990	10
				Benign	12	14087

Table 11. SVM Classifier Performance Using 16 Features

Accuracy	Precision	Sensitivity	Specificity	Confusion Matrix		
					Malicious	Benign
99.90	99.87	99.89	99.90	Malicious	9987	13
				Benign	11	14085

Table 12. SVM Classifier Rules Using 16 Features

Class	Malicious	Benign	Minimised
Rules	33,589	31,947	8,043

4.3.1 Training k-NN Classifiers A k-NN classifier was trained using the same 34 features as for the NN and SVM classifiers. Here (unlike in previous work [30]) the k parameter was optimised to be 4. Table 15 shows the performance results for k-NN when using 34 features.

Again, a k-NN classifier trained using 16 features was also developed and the results of testing this can be seen in Table 16. As for the other base classifiers, the Boolean function that this classifier defines can be extracted and the number of rules that the gives can be seen in Table 17.

4.3.2 Rule Extraction As in the previous sections, Boolean rule extraction was performed by sampling the 34 feature k-NN classifier using the 32/64/128 sample tactic. The results are presented in Table 18. The number of rules extracted are given in Table 19.

4.4 Timings

A major motivation for investigating the number of samples required to extract good Boolean rules from other classifiers is the time taken by the rule extraction from [29]. In this section, the time taken for rule extraction using the 32/64/128 tactic is given for NN, SVM and kNN. These timings are given in Table 20.

Table 13. Results for Rules Extracted from SVM using 32/64/128 Samples

	Accuracy	Precision	Sensitivity	Specificity	Confusion Matrix		
1 Feature	91.96	80.70	99.92	87.95		Malicious	Benign
					Malicious	8070	1930
					Benign	6	14090
2 Features	91.96	80.70	99.92	87.95		Malicious	Benign
					Malicious	8070	1930
					Benign	6	14090
4 Features	98.74	97.03	99.93	97.93		Malicious	Benign
					Malicious	9703	297
					Benign	6	14091
8 Features	98.24	95.80	99.97	97.10		Malicious	Benign
					Malicious	9580	420
					Benign	2	14094
10 Features	99.18	98.07	99.96	98.64		Malicious	Benign
					Malicious	9807	193
					Benign	3	14093
12 Features	99.09	97.85	99.97	98.49		Malicious	Benign
					Malicious	9785	215
					Benign	2	14094
16 Features	99.90	99.88	99.89	99.91		Malicious	Benign
					Malicious	9988	12
					Benign	11	14085

Table 14. Numbers of SVM Rules Using 32/64/128 Samples

Features	Malicious	Benign	Minimised
1 Feature	1	1	1
2 Features	2	2	1
4 Features	8	8	4
8 Features	79	177	35
10 Features	327	697	129
12 Features	1,193	2,903	448
16 Features	35,780	29,756	5,884

4.5 Labelling Via Sampling

The core of the rule extraction methods presented in this paper is sampling a classifier to give an assignment of a Boolean value to a row in a truth table. This section investigates the sampling and how clean a division is achieved. Figures 1, 2, 3 present results on this for NN, SVM and k-NN respectively. Each figure gives two results, developed by considering rule extraction for 16 features using a fixed sample size of 32 (note that this differs from the 32/64/128 tactic used in the earlier results, although the performance

Table 15. k-NN Classifier Performance Using 34 Features

Accuracy	Precision	Sensitivity	Specificity	Confusion Matrix		
					Malicious	Benign
99.68	100	99.25	100	Malicious	1000	0
				Benign	75	14021

Table 16. k-NN Classifier Performance Using 16 Features

Accuracy	Precision	Sensitivity	Specificity	Confusion Matrix		
					Malicious	Benign
99.83	99.99	99.61	99.99	Malicious	9999	1
				Benign	39	14057

Table 17. k-NN Classifier Rules Using 16 Features

Class	Malicious	Benign	Minimised
Rules	47,244	18,292	1,598

of the extracted classifiers is not dramatically different). The first result (in blue) details how many rows of the truth table (with 2^{16} rows) result from the number of cases; that is, Iterations gives the number of rows which has a split where Cases is the number of samples giving malicious. The second result (in orange) gives the number of testing instances that are assigned from a row of the truth table whose value was determined by a sample with Cases being the number of samples giving malicious (for example, if a test instance is determined to be malicious, Cases is the number of samples that have malicious, say 30).

5 Discussion

As has already been established in [30, 31] a variety of machine learning classifiers can be applied to the XSS detection problem giving very good results. This is confirmed here in Tables 2, 10 and 15 giving performance results for NN, SVM and k-NN classifiers trained using the 34 highest ranked features. These results are accompanied by Tables 3, 11 and 16 giving the results when training NN, SVM and k-NN classifiers using only the 16 highest ranked features, which again give good results, if (particularly for NN) not quite as good as for the higher number of features.

Each of the classifiers above defines a Boolean function, which might be used to replace the original classifier. For the 16 feature classifiers, this Boolean function can be calculated directly, but for the 34 feature classifiers this is computationally intractable. The first question posed by this paper, is whether it is possible to find useful approxi-

Table 18. Results for Rules Extracted from k-NN using 32/64/128 Samples

	Accuracy	Precision	Sensitivity	Specificity	Confusion Matrix		
1 Feature	58.49	0	0	58.49		Malicious	Benign
					Malicious	0	10000
					Benign	0	14096
2 Features	98.95	97.60	99.87	98.32		Malicious	Benign
					Malicious	9760	240
					Benign	12	14084
4 Features	98.74	97.08	99.88	97.96		Malicious	Benign
					Malicious	9708	292
					Benign	11	14085
8 Features	99.48	98.25	92.78	99.44		Malicious	Benign
					Malicious	9925	75
					Benign	772	13324
10 Features	96.50	96.91	94.78	97.77		Malicious	Benign
					Malicious	9691	309
					Benign	533	13563
12 Features	99.72	99.79	99.54	99.85		Malicious	Benign
					Malicious	9979	21
					Benign	46	14050
16 Features	97.06	99.95	93.43	99.96		Malicious	Benign
					Malicious	9995	5
					Benign	702	13394

Table 19. Numbers of k-NN Rules Using 32/64/128 Samples

Features	Malicious	Benign	Minimised
1 Feature	1	1	1
2 Features	3	1	1
4 Features	7	9	5
8 Features	151	105	32
10 Features	593	431	105
12 Features	3,334	762	94
16 Features	44,120	21,416	2,843

mations to the Boolean functions described by the higher dimensional classifiers? The existence of the 16 feature Boolean classifiers suggests that it is. Table 5 repeats the experiments from [29] using the sampling based approach to find a series of Boolean classifiers using an increasing number of features to approximate the 34 feature NN classifier. The performance of the approximating Boolean classifier using 16 features matches (in fact, slightly betters) that of the neural network that it is modelling, with 99.90% accuracy and 99.94% precision, demonstrating that rule extraction has been

Table 20. Timing of Rule Extraction from NN/SVM/k-NN Classifiers Using 32/64/128 Samples

Features	NN	SVM	k-NN
1 Feature	1 sec	1 sec	1 sec
2 Features	2 sec	1 sec	1 sec
4 Features	3 sec	1 sec	6 sec
8 Features	57 sec	9 sec	101 sec
10 Features	212 sec	36 sec	419 sec
12 Features	755 sec	135 sec	2,016 sec
16 Features	10,645 sec	3,720 sec	23,271 sec

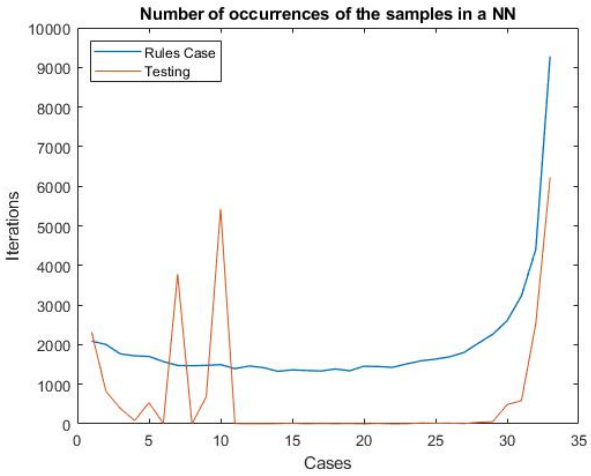

Fig. 1. Number of occurrences of the samples in a NN

successfully accomplished. For comparison, the extracted rule-based Boolean function classifier in Table 5 performs slightly better than those for the 16-feature neural network in Table 3. These results show that rule extraction works, but the sampling used, where 1024 samples were used to generate each row of the truth table for the Boolean function, is slow. As can be seen in Table 7 the time taken to build the Boolean functions increases exponentially, with the best approximation using 16 features taking more than five days of computation.

The second question posed by this paper is whether the number of samples can be reduced (speeding up the approximation process) whilst maintaining precision. This is answered by Table 8 where a small number (32) of samples is used to determine the value of a row in the truth table, and increasing this (to 64, then 128 samples) only when the sample does not give a clear cut answer. The results show that the performance of the resulting Boolean classifiers is comparable to that of the Boolean classifiers extracted by using a larger number of samples, hence also comparable to that of the underlying

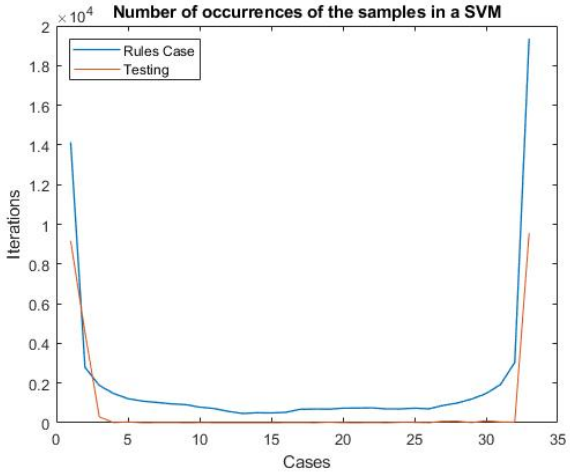

Fig. 2. Number of occurrences of the samples in a SVM

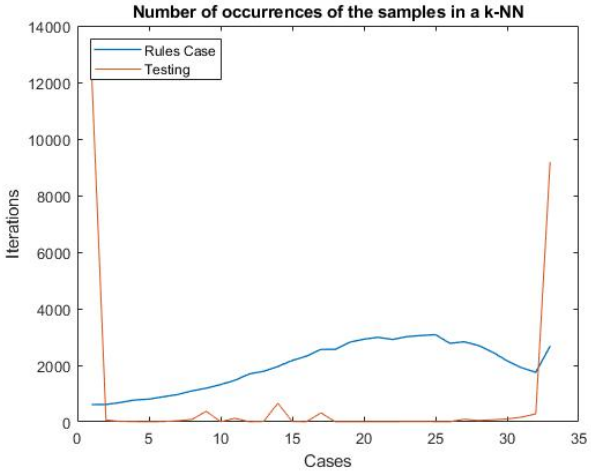

Fig. 3. Number of occurrences of the samples in a k-NN

neural network when using 16 features. Table 20 shows that this approach is indeed faster, taking about 3 hours of computation.

Third question is whether this approach works for classifiers developed using other machine learning techniques? This work considers: i) support vector machines that to some extent share the problem of neural networks in that the output of learning is a function which is hard to interpret, ii) k-nearest neighbour where the interpretability of the output is clear, but where the output model is large, being essentially a representation of the training data. Tables 13 and 18 give the results of repeating the approach

to approximating neural network classifiers using instead SVM and k-NN respectively. The results confirm that the approach works for other classifiers. The performance of the Boolean functions extracted from the 34 feature SVM are excellent, with the 16 feature Boolean classifier working as well as the underlying base SVM. The performance of the Boolean functions extracted from the 34 feature k-NN classifier, whilst giving performance of over 90%, are not quite as strong as for the NN and SVM classifiers. In particular the 16 feature Boolean function performs noticably less well that the k-NN classifier it is approximating (and the 12 feature Boolean function). The authors believe that this is because NN and SVM classifiers are learning functions that discriminate between the features, weighting some more heavily than others; in contrast, k-NN classifiers simply measure distance from the training data, with no feature being more important than others.

When performing rule extraction, a series of approximations using an increasing number of features have been built and evaluated in Tables 8, 13 and 18, as well as Table 5. The number of rules both before and after minimisation are given in Tables 6, 9, 14, and 19. As might be expected, as the number of features increases, the number of rules (after minimisation) increases too, and the performance of the resulting classifiers improves. The improvements are not necessarily monotonic, but the pattern is clear. If comparing against the Boolean function extracted from the 16 feature classifers, the number of minimised rules is comparable (except for SVM).

The fourth question addressed is how well the sampling method gives Boolean values, that is, do all the samples give the same value, or do the samples give a split decision? This is considered in Figures 1, 2, 3. The first result (the blue line) plots the sample split (Cases) against the number of rows of the 16 feature truth table which derive from this split. As can be seen in Figure 2, SVM gives the cleanest split with the value for most rows deriving from a sample with only a handful of sample instances not agreeing with the final label. Figure 1 shows a similar pattern for the neural network, although this is shallower with more cases where the number of samples for each value are close to each other. Figure 3, however, gives a different pattern for k-NN, with fewer clean cut cases, hinting at why rule extraction works less well. The second result (the orange line) plots the number of tests that were labelled using a rule whose split is given by Cases. Here it can be seen why the approach works so well. For test data (which is not a chimera of a row of the truth table and a sample from the training data) almost all of the values assigned come from rules whose value comes from a clean cut sample. This applies to all three base classifiers. That is, the doubt in the sampling comes from examples which do not occur in practice.

The final question is what level of explainable AI has been extracted from classifiers in the form of the Boolean rule-based systems? The approximations described in this work give classifiers whose reasoning can be described, allowing decision making to be auditable. The successive approximations show that relatively good performance can be achieved with the use of only a small number of features. That the sampling approach gives approximations with some degree of noise is illustrated across the tabled results where anomalous cases can be found. For example, the 8 feature case in Table 5, where the introduction of feature 7, URL addresses, leads to some additional misclassifications, compared to the courser 4 feature classifier. It should also be noted that the very

course 1 and 2 feature classifiers still give useful result, with all the 2 feature classifiers giving over 90% precision. The reason for this result is that the highest ranking feature is the use of "Alert" within the script and that a high proportion of attacks in the database use this, whilst it is rarely used in benign scripts. This first feature is very powerful. This observation (whilst not surprising to the authors) is a good illustration of XAI in action, where the rule-based system has made the explanation explicit.

The best approximation still requires thousands of rules even after minimisation. It is not clear that each individual decision can be interpreted by a human user, in the context of the larger number of rules. However it should be noted that these rules are simply the flattening of a single binary decision-tree over 16 features. It seems unlikely that a smaller datastructure can be used successfully as a classifier. In either view, the extracted rules mean that decision making is always auditable, the reasoning for any decision can be traced.

As noted in the methodology, the current approach requires a double use of the training set, firstly to train the classifiers, and secondly to guide the sampling approach used in the approximation of the classifiers by Boolean functions. However, given the size of the Boolean function described by the trained classifiers, some kind of guidance seems inevitable in a black box approach to approximation. The black box approach has worked, resulting in successfully extracting rules in form of $(if...then...else)$ in order to distinguish malicious and benign scripts without delving deeper into the inner structure of the classifiers.

6 Conclusion

This paper develops the approach to rule extraction first described in [29]. It considers machine learning for classification problems where the feature space is Boolean, and the classes are also Boolean. The example in this work is the classification of JavaScript as malicious (an XSS attack) or benign, where the actual script is abstracted to a Boolean feature set. The rule extraction first finds a Boolean function as a truth table describing the classifier, then simplifies this. The Boolean function might be an exact description of the classifier, or an approximation built using a sampling technique.

Rule extraction is successfully demonstrated for three kinds of classifier trained to detect XSS attacks: neural networks, support vector machines and k-nearest neighbour. Approximations to the full 34 feature classifiers were considered at different levels of granularity. The most precise approximations are over 16 features. This work shows how the sampling technique first suggested in [29] can be adapted to work with a reduced number of samples, producing approximations much more quickly. Using this new, faster, approach to sampling, the 16 feature approximation to the original neural network gives 99.87% accuracy and 99.86% precision. These results are as good as those for the initial classifier and can be computed relatively quickly, extending and improving upon the results in [29]. Similar results are obtained for SVM, and good, though less reliable results for k-NN.

The number of rules extracted grows with the number of features used in the approximation. As discussed in Section 5, this means that these rules are auditable – the reasoning for any given classification can easily be looked up and interpreted – and es-

sentially form a 16 feature binary decision tree which, while relatively large, should be seen as an explainable classifier.

Future work is to investigate how this approach might be generalised to features which are not Boolean valued, by piecewise approximation, or otherwise. Alternative ways of computing the rules (perhaps using BDDs), and further approximation to give more compact rules sets will also be explored.

In conclusion, XAI principles have been followed to give a procedure resulting in explanations of black box classifiers as a set of Boolean rules which can be understood by human users, leading to successful rule extraction.

References

1. Ardiansyah, S., Majid, M.A., Zain, J.M.: Knowledge of extraction from trained neural network by using decision tree. In: 2nd International Conference on Science in Information Technology (ICSITech). pp. 220–225. IEEE (2016)
2. Augasta, M.G., Kathirvalavakumar, T.: Rule extraction from neural networks—a comparative study. In: International Conference on Pattern Recognition, Informatics and Medical Engineering (PRIME-2012). pp. 404–408. IEEE (2012)
3. Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.R., Samek, W.: On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation. PLoS ONE **10**(7), e0130140 (2015)
4. Baehrens, D., Schroeter, T., Harmeling, S., Kawanabe, M., Hansen, K., Müller, K.R.: How to Explain Individual Classification Decisions. Journal of Machine Learning Research **11**, 1803–1831 (2010)
5. Baesens, B., Martens, D., Setiono, R., Zurada, J.M.: Guest Editorial White Box Nonlinear Prediction Models. IEEE Transactions on Neural Networks **22**(12), 2406–2408 (2011)
6. Bondarenko, A., Aleksejeva, L., Jumutc, V., Borisov, A.: Classification Tree Extraction from Trained Artificial Neural Networks. Procedia Computer Science **104**, 556–563 (2017)
7. Breiman, L., et al.: Statistical modeling: The two cultures (with comments and a rejoinder by the author). Statistical Science **16**(3), 199–231 (2001)
8. Bryant, R.E.: Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams. ACM Computing Surveys **24**, 293–318 (1992)
9. Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In: Empirical Methods in Natural Language Processing. p. 1724–1734. Association for Computational Linguistics (2014)
10. Craven, M.W., Shavlik, J.W.: Using Sampling and Queries to Extract Rules from Trained Neural Networks. In: Proceedings of the Eleventh International Conference on Machine Learning, pp. 37–45. Morgan Kaufmann (1994)
11. Craven, M.W., Shavlik, J.W.: Extracting tree-structured tepresentations of trained networks. In: Advances in Neural Information Processing Systems. pp. 24–30. MIT Press (1996)
12. Dancey, D., McLean, D.A., Bandar, Z.A.: Decision tree extraction from trained neural networks. In: Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference. pp. 515–519. AAAI Press (2004)
13. Etchells, T.A., Lisboa, P.J.: Orthogonal search-based rule extraction (OSRE) for trained neural networks: a practical and efficient approach. IEEE Transactions on Neural Networks **17**(2), 374–384 (2006)
14. Giménez, C.T., Villegas, A.P., Marañón, G.Á.: HTTP data set CSIC 2010. Information Security Institute of CSIC (Spanish Research National Council) (2010)

15. Gunning, D.: Explainable Artificial Intelligence (XAI). Tech. Rep. DARPA/I20, Defense Advanced Research Projects Agency (2016)
16. Hailesilassie, T.: Rule extraction algorithm for deep neural networks: A review. arXiv preprint arXiv:1610.05267 (2016)
17. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Computer Vision and Pattern Recognition. pp. 770–778. IEEE (2016)
18. Huysmans, J., Baesens, B., Vanthienen, J.: Using rule extraction to improve the comprehensibility of predictive models. Tech. Rep. KBI 0612, Katholieke Universiteit Leuven (2006)
19. Jain, T.K., Kushwaha, D.S., Misra, A.K.: Optimization of the Quine-McCluskey Method for the Minimization of the Boolean Expressions. In: Fourth International Conference on Autonomic and Autonomous Systems. pp. 165–168. IEEE Computer Society (2008)
20. Julian, K.D., Lopez, J., Brush, J.S., Owen, M.P., Kochenderfer, M.J.: Policy compression for aircraft collision avoidance systems. In: Proceedings of the 35th Digital Avionics Systems Conference . pp. 1–10. IEEE Press (2016)
21. Karnaugh, M.: The map method for synthesis of combinational logic circuits. Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics **72**, 593–599 (1953)
22. Katz, G., Barrett, C., Dill, D., Julian, K., Kochenderfer, M.: Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In: Computer Aided Verification. Lecture Notes in Computer Science, vol. 10426, pp. 97–117. Springer (2017)
23. Keedwell, E., Narayanan, A., Savic, D.: Creating rules from trained neural networks using genetic algorithms. Internation Journal of Computers, System and Signal **1**(1), 30–42 (2000)
24. Likarish, P., Jung, E., Jo, I.: Obfuscated malicious Javascript detection using classification techniques. In: Malicious and Unwanted Software (MALWARE). pp. 47–54. IEEE (2009)
25. Mahendran, A., Vedaldi, A.: Understanding Deep Image Representations by Inverting Them. In: Computer Vision and Pattern Recognition. pp. 5188–5196. IEEE (2015)
26. Manojlovic, V.: Minimization of Switching Functions using Quine-McCluskey Method. International Journal of Computer Applications **82**(4), 12–16 (2013)
27. Martens, D., Baesens, B., Van Gestel, T.: Decompositional Rule Extraction from Support Vector Machines by Active Learning. IEEE Transactions on Knowledge and Data Engineering **21**(2), 178–191 (2009)
28. MathWorks: Feature selection. https://uk.mathworks.com/help/stats/feature-selection.html (2019), accessed: 11/3/2019
29. Mereani, F.A., Howe, J.M.: Exact and Approximate Rule Extraction from Neural Networks with Boolean Features. In: Proceedings of the 11th International Joint Conference on Computational Intelligence. pp. 424–433. ScitePress (2019)
30. Mereani, F.A., Howe, J.M.: Detecting Cross-Site Scripting Attacks Using Machine Learning. In: International Conference on Advanced Intelligent Systems and Computing, vol. 723, pp. 200–210. Springer (2018)
31. Mereani, F.A., Howe, J.M.: Preventing Cross-Site Scripting Attacks by Combining Classifiers. In: Proceedings of the 10th International Joint Conference on Computational Intelligence - Volume 1. pp. 135–143. ScitePress (2018)
32. Nadji, Y., Saxena, P., Song, D.: Document Structure Integrity: A Robust Basis for Cross-site Scripting Defense. In: Network and Distributed System Security Symposium. Internet Society (2009)
33. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th International Conference on Machine Learning. pp. 807–814. Omnipress (2010)
34. Nguyen, A., Yosinski, J., Clune, J.: Multifaceted Feature Visualization: Uncovering the Different Types of Features Learned by Each Neuron in Deep Neural Networks. arXiv preprint arXiv:1602.03616 (2016)

35. OWASP Top 10 - 2017 rc1 (2017), https://www.owasp.org. Accessed: 7/6/2017
36. Rhode, M., Burnap, P., Jones, K.: Early Stage Malware Prediction Using Recurrent Neural Networks. Computers and Security **77**, 578–594 (2017)
37. Rickmann, S.: Logic Friday (version 1.1.4)[computer software]. https://web.archive.org/web/20131022021257/http://www.sontrak.com/ (2012), accessed: 24/11/2018
38. Rudell, R.L.: Multiple-valued logic minimization for PLA synthesis. Tech. Rep. UCB/ERL M86/65, University of California, Berkeley (1986)
39. Saad, E.W., Wunsch II, D.C.: Neural network explanation using inversion. Neural networks **20**(1), 78–93 (2007)
40. Saito, K., Nakano, R.: Medical diagnostic expert system based on PDP model. In: Proceedings of IEEE International Conference on Neural Networks. vol. 1, pp. 255–262 (1988)
41. Schmitz, G.P., Aldrich, C., Gouws, F.S.: ANN-DT: an algorithm for extraction of decision trees from artificial neural networks. IEEE Transactions on Neural Networks **10**(6), 1392–1401 (1999)
42. Schwender, H.: Minimization of Boolean Expressions using Matrix Algebra. Tech. rep., Technical Report//Sonderforschungsbereich 475, Komplexitätsreduktion in Multivariaten Datenstrukturen, Universität Dortmund (2007)
43. Setiono, R., Liu, H.: Understanding Neural Networks via Rule Extraction. In: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence. vol. 1, pp. 480–485. Morgan Kaufmann (1995)
44. Setiono, R., Liu, H.: NeuroLinear: From neural networks to oblique decision rules. Neurocomputing **17**(1), 1–24 (1997)
45. Simonyan, K., Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition. In: International Conference on Learning Representations (2015)
46. Taha, I., Ghosh, J.: Three techniques for extracting rules from feedforward networks. In: Intelligent Engineering Systems Through Artificial Neural Networks. pp. 23–28. ASME Press (1996)
47. Thrun, S.B.: Extracting Provably Correct Rules from Artificial Neural Networks. Tech. rep., University of Bonn (1993)
48. Tsukimoto, H.: Extracting rules from trained neural networks. IEEE Transactions on Neural Networks **11**(2), 377–389 (2000)
49. Wang, H., Lu, Y., Zhai, C.: Latent Aspect Rating Analysis Without Aspect Keyword Supervision. In: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 618–626. KDD '11, ACM (2011)
50. Wang, W.H., Lv, Y.J., Chen, H.B., Fang, Z.L.: A Static Malicious JavaScript Detection Using SVM. In: Proceedings of the International Conference on Computer Science and Electronics Engineering. vol. 40, pp. 21–30 (2013)
51. Weinberger, J., Saxena, P., Akhawe, D., Finifter, M., Shin, R., Song, D.: A Systematic Analysis of XSS Sanitization in Web Application Frameworks. In: European Symposium on Research in Computer Security. Lecture Notes in Computer Science, vol. 6879, pp. 150–171. Springer (2011)