IEEE TRANSACTIONS ON
EMERGING TOPICS
IN COMPUTING

Received 15 October 2013; revised 13 March 2014; accepted 16 March 2014. Date of publication 29 April 2014;
date of current version 30 October 2014.

Digital Object Identifier 10.1109/TETC.2014.2316510

Large Iterative Multitier Ensemble Classifiers for Security of Big Data

JEMAL H. ABAWAJY, (Senior Member, IEEE), ANDREI KELAREV, AND MORSHED CHOWDHURY

School of Information Technology, Deakin University, Burwood 3125, Australia

CORRESPONDING AUTHOR: J. ABAWAJY (jemal@deakin.edu.au)

ABSTRACT This paper introduces and investigates large iterative multitier ensemble (LIME) classifiers specifically tailored for big data. These classifiers are very large, but are quite easy to generate and use. They can be so large that it makes sense to use them only for big data. They are generated automatically as a result of several iterations in applying ensemble meta classifiers. They incorporate diverse ensemble meta classifiers into several tiers simultaneously and combine them into one automatically generated iterative system so that many ensemble meta classifiers function as integral parts of other ensemble meta classifiers at higher tiers. In this paper, we carry out a comprehensive investigation of the performance of LIME classifiers for a problem concerning security of big data. Our experiments compare LIME classifiers with various base classifiers and standard ordinary ensemble meta classifiers. The results obtained demonstrate that LIME classifiers can significantly increase the accuracy of classifications. LIME classifiers performed better than the base classifiers and standard ensemble meta classifiers.

INDEX TERMS LIME classifiers, ensemble meta classifiers, random forest, big data.

I. INTRODUCTION

Big Data has become ubiquitous and crucial for numerous application domains, thereby leading to signifiable challenges from the point of view of data management perspective [1], [2]. It has become particularly important in view of the rapid growth of Cloud services [3]–[5]. The development and expansion of the Cloud creates new opportunities for the users and requires further research to address novel tasks and requirements [6]–[8]. It is important not only to invent new methods tackling these tasks, but also to investigate ways of adapting previous techniques well known in related areas such as grid computing [9], [10]. Security has been one of the major issues required for the use of Big Data. Let us refer to [11] for an overview and analysis of the top ten Big Data security and privacy challenges. The scientific challenges faced by the Cloud computing security have been discussed in the survey articles [12], [13] and have been considered also in [14]–[17].

This article introduces four-tier Large Iterative Multitier Ensemble (LIME) classifiers specifically designed for applications concerning the information security of Big Data and generated as explained in Section III. The investigation of this new construction is important, because the role of algorithms for analysis of Big Data has been growing. As a particular application direction for experiments we address the problem of malware detection, which is essential for the security of Big Data.

The main aim of this paper is to develop LIME classifiers as a general technique that may be useful for the analysis of Big Data in various application domains. If a dataset is not large enough, then the LIME classifier will revert to using only a base classifier or just a small part of the whole system and will not improve the quality of the classification. We carry out a systematic experimental investigation of the performance of LIME classifiers for a problem concerning security of Big Data. This new iterative construction is illustrated in Figure 1. LIME classifiers can combine diverse ensemble meta classifiers into one iterative hierarchical system and can be very large.

The term malware refers to malicious software or to malicious computer programs [18], [19]. Malware has been an ever growing threat to security for a long time. Malware can

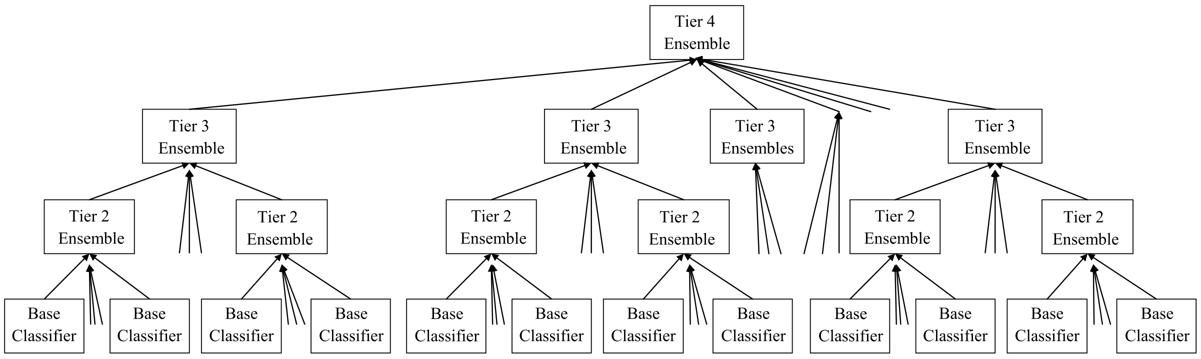


FIGURE 1. **Four-tier LIME classifier processing big data. The direction of arrows indicates data flow.**

infect computers and information repositories via networks, attached hardware devices or over the Internet from websites or email. Here let us only briefly refer to [20]–[25] for background information and preliminaries on malware. More details are given in Section II.

Malware detection represents a significant challenge for security of Big Data. Recent explosion in malware proliferation is explained not only by the continued growth of all traditional avenues for its propagation, but also by several new factors.

Firstly, mobile devices have become capable of running software of complexity comparable to that of desktop machines. This has dramatically increased the number of devices vulnerable to malware. At present, the operating systems of personal digital assistants and smart phones can execute and spread sophisticated malware similar to that affecting only personal computers previously. Nowadays malware can exploit new means of propagation via Bluetooth, short message service, and multimedia messaging service. However, the use of anti-malware for smart phones is hindered by the available battery energy. The existing anti-malware systems have not been adapted to smart phones yet.

Secondly, the competition between malware writers on the one side versus the internet security research combined with software vendors creating anti-malware systems on the other side has lead to the development of advanced obfuscation techniques and readily available programs that can be used to modify existing malware making it unrecognisable. Novel behaviour analysis methods using dynamic features have been investigated to counteract malware obfuscation, see Section II.

However, it is also quite conceivable that malware writes will be able to conceal various behavioural features of malware equally well, as the creators of Trojans have been able to do. A universal and effective method for concealing behaviour was introduced and studied in [26]. Besides, the main drawback of behaviour analysis is that it can be applied in practical situations to detect malware only once the latter has been executed, which means that it has already had a chance to perform its malicious function and spread further possibly in a new modified form.

Finally, the rapid expansion of cloud service providers and social networks has created a multitude of new communication avenues that potentially can be used for malware propagation.

These circumstances have lead to enormous variety in the number of instances and different incomparable and unrecognisable versions of malware being created daily and propagating over the Internet. Now researchers have to investigate and develop new methods for malware analysis and detection that are more suitable for Big Data.

In this paper we use LIME classifiers with four tiers for the detection of malware using Big Data. LIME classifiers are automatically generated as iterative multitier ensembles of ensembles. The initialization and generation stages of a LIME classifier are explained in Section III, where it is also shown how to aggregate the classifiers at different levels to obtain the LIME classifier.

Our experimental results show that four-tier LIME classifiers achieved substantially better performance in comparison with the base classifiers or standard ensemble meta classifier classifiers. This demonstrates that our new method of combining diverse ensemble meta classifiers into one unified four-tier ensemble incorporating diverse ensemble meta classifiers as parts of other ensemble meta classifiers can be applied to improve classifications.

The paper is organised as follows. Section II contains a brief overview of previous related work. Section III describes four tier LIME classifiers investigated in this paper. Section VI is devoted to preprocessing of data. Sections IV and V deal with the base classifiers and standard ensemble meta classifiers used in our experiments and included in the LIME classifiers. Section VII presents the outcomes of experiments comparing the effectiveness of base classifiers, standard ensemble meta classifiers and four-tier LIME classifiers. These results are discussed in Section VIII. Main conclusions are presented in Section IX.

II. RELATED WORK

Major security challenges facing the analysis of Big Data and the Cloud have been considered in [11]–[13].

Researchers in [1] deal with the problems motivated by the unprecedented growth in the amount of available data, as well as with the new opportunities presented by the development of the Cloud. It big data challenges including big data diversity, big data reduction, big data integration and cleaning, big data indexing and query, and finally big data analysis and mining.

The paper [14] is looking at a cloud data storage preventing harvesting. The problem concerns the cloud customers, the cloud business centre which provides services, and the cloud data storage centre. Data stored in the data storage centre comes from a variety of customers and some of these customers may compete with each other in the market place or may own data which comprises confidential information about their own clients. Cloud staff have access to data in the data storage centre which could be used to steal identities or to compromise cloud customers. The paper proposes an efficient method of data storage which prevents staff from accessing data which can be abused as described above. We also suggest a method of securing access to data which requires more than one staff member to access it at any given time. This ensures that, in case of a dispute, a staff member always has a witness to the fact that she accessed data.

For general background information on the methodology, systems and applications of the Cloud Computing, the associated Quality of Service Management, the design of Cloud Workflow Systems, and tradeoffs between the computation and data storage we refer to [4]–[6], respectively.

The paper [7] investigates an iterative hierarchical key exchange scheme for secure scheduling of big data applications in cloud computing. The privacy preservation over big data on cloud is considered in [17].

In [27] the authors present a scalable, automated approach for detecting and classifying malware by using pattern recognition algorithms and statistical methods. The proposed framework combines the static features of function length and printable string information extracted from malware samples into a single test.

In [24] the authors compare two disparate sets of malware collected in different time periods and train an antivirus strategy on the earlier set to determine how well it will manage to handle the later set. An anti-virus strategy is used that integrates dynamic and static features extracted from the executables to to detect malware versus cleanware and to classify malware by distinguishing between malware families. This paper provides strong evidence that anti-virus techniques which work well on malware developed at a certain time may continue to be effective on malware developed at a much later time.

Feature reduction based on Information Gain was used in [25] to speed up the processes of the classification of malware and the identification of malware from a set combined with cleanware. The number of features was reduced from 7,605 to just over 1,000. The lead to a reduction in false negative rates by a factor of about 1/3. The speed of running the tests improved by a factor of approximately 3/5. A small loss in accuracy was observed. However, the improved false negative rate along with significant improvement in speed demonstrated that feature reduction can be pursued as a tool to prevent algorithms from becoming intractable due to too much data.

The first classification method integrating static and dynamic features into a single test was presented in [20]. The approach proposed there improved on previous results using individual features collected separately. The time required for the test was reduced by half. Robustness to changes in malware development was tested. It was shown that to achieve acceptable accuracy in classifying the latest malware, some older malware should be included in the set of data.

A lightweight malware detection system for detecting, analysing and predicting malware propagating via SMS and MMS messages on mobile devices is proposed in [28]. It deploys agents in the form of hidden contacts on the device to capture messages sent from malicious applications. The captured messages are fed to a latent space model, for analysis to estimate the current dynamics and predict the future state of malware propagation within the mobility network.

A scalable system for network-level behavioural clustering of HTTP-based malware is presented in [29]. It groups newly collected malware samples into malware family clusters. This aim of creating clusters is to facilitate the generation of high quality network signatures for detecting botnet communications at the network perimeter. The scalability of the clustering system is achieved by using a simplified multi-step clustering process incorporating incremental clustering algorithms that run efficiently on very large datasets. The system reduces processing times and scales well to large datasets containing tens of thousands of distinct malware samples.

Frequency of the appearance of opcode sequences is used in [30] to prepare data for data mining algorithms trained to detect malware.

A concept of genetic footprint is proposed in [31]. It can be used to detect malicious processes at run time. The genetic footprint consists of selected parameters maintained inside the PCB of a kernel for each running process. It defines the semantics and behaviour of an executing process.

A graph-based method to detect unknown malware is presented in [32]. It uses the function call graph of an executable, which includes the functions and the call relations between them. The features are defined according to both the statistical information and the topology of the function call graph.

The article [33] studied the proactivity of malware detection using ten different Perceptron derived algorithms. The proactivity score was defined by

$$\text{ProactivityScore} = \frac{TP}{TP+FN+\frac{1}{k}FP}$$

where TP is the number of true positives, FN is the number of false negatives, FP is the number of false positives, and k was taken equal to 0.5%.

III. LIME CLASSIFIERS

A number of methods for creating ensemble classifiers are well known in artificial intelligence and data mining.

Every ensemble meta classifier combines a collection of base classifiers into a common classification system. Efficient multi-tier classifiers and more general multi-classifier systems have been explored, for example, in the previous publications [22], [23], [34]–[37]. Our construction of four-tier LIME classifiers was inspired by previous research in the literature, but is different.

Traditional ensemble meta classifiers generate their collection of base classifiers given an indication, or an example, or a template of only one base classifier as an input parameter. After the generation stage, they use the whole ensemble of the base classifiers to process data, collect their outputs and combine them to prepare the final decision. For example, Random Forest automatically generates a collection of Random Trees and uses them as shown in Figure 2. Many other known ensemble meta classifiers function in a similar fashion, using other base classifiers and different techniques for generating and combining them.

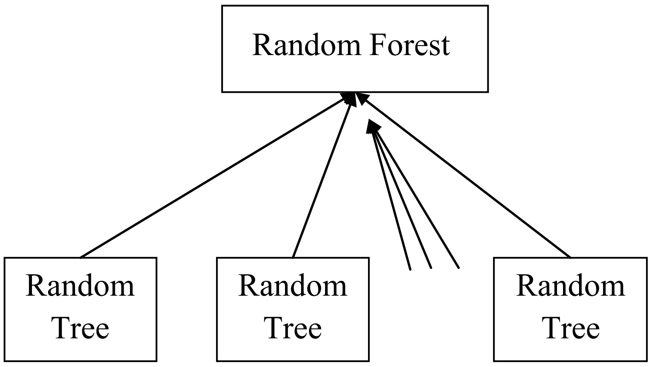


FIGURE 2. Random forest is a two-tier ensemble classifier based on Random trees.

The present article is devoted to a new scheme that creates very large LIME classifiers tailored for handling Big Data. LIME classifiers automate the process of generating a large multitier system. They make it easy to generate very large classifiers combining diverse ensemble meta classifier methods at several levels.

LIME classifiers are used with four tiers in this paper. They incorporate diverse ensemble meta classifiers into second, third and fourth tiers simultaneously and combine them into one integrated iterative system so that third tier ensemble meta classifiers acts as an integral part of the fourth tier ensemble meta classifier, and each second tier ensemble meta classifier is an integral part of its third tier ensemble meta classifier parent, as shown in Figure 1. The fourth tier ensemble meta classifier of this construction invokes third tier ensemble meta classifiers, and in turn they invoke their second tier ensemble meta classifiers in an iterative fashion. The present article is devoted to experiments comparing the performance of four-tier LIME classifiers, their base classifiers and standard ensemble meta classifiers in a special problem concerning security of Big Data.

It is easy to set up and generate a LIME classifier. All third tier ensemble meta classifiers are generated by the fourth tier ensemble meta classifier given just one instance of a second tier ensemble as an input parameter for the generation stage. The forth tier ensemble meta classifier generates all third tier ensemble meta classifiers and executes them in exactly the same way as it usually handles base classifiers. Similarly, each third tier ensemble meta classifier applies its method to generate and combine its second tier ensemble meta classifiers. Finally, the second tier ensemble meta classifiers generate, execute and combine their base classifiers in their standard fashion.

To start the process a designer has to initialize a four-tier LIME classifier by specifying which ensemble meta classifier will operate at the fourth tier. Then the designer provides a parameter to the fourth tier ensemble meta classifier indicating, which third tier ensemble meta classifier is to be used as a part of the standard generation process of the fourth tier ensemble meta classifier. After that, the designer specifies the second tier ensemble meta classifier method to be used by the third tier ensemble meta classifier, and the base classifier handled by the second tier ensemble meta classifier. The initialization step is shown in Figure 3.

In this paper we used diverse ensemble meta classifiers and base classifiers implemented in the Waikato Environment for Knowledge Analysis (WEKA). All options chosen by a designer for a LIME classifier can be indicated in the WEKA SimpleCLI command line as shown in Figure 5. Then the whole system is generated automatically by the SimpleCLI, using the embedded iterative and recursive capability of Java programming.

After initialization each of the ensemble meta classifiers chosen by the designer uses its own method of generating the classifiers at the lower tier. First, the fourth level ensemble meta classifier generates a collection of the classifiers at the third tier as shown in Figure 4. Second, each of the third tier ensemble meta classifiers created in Figure 4 applies its own scheme of generating second tier ensemble meta classifiers as shown in Figure 6.

Finally, as illustrated in Figure 6, each of the second tier ensemble meta classifiers, produced at the preceding stage as in Figure 6, uses its method of generating a collection of base classifiers according to the type of the base classifier indicated in Figure 3. This concludes the generation stage of work of the LIME classifier.

After that the LIME classifier processes data as shown in Figure 1, where the direction of arrows indicates data flow. Edges not connected to classifiers indicate the direction of possible data flow from many more classifiers that are not depicted in the diagram.

The base classifiers analyse the features of the original instances and pass on their output to the second tier ensemble meta classifiers. The second tier ensemble meta classifiers collect all outputs of the base classifiers, combine them, and send their own output to their parent third tier ensemble meta classifiers. Likewise, the third tier ensemble meta classifiers collect the outputs of the second tier ensemble meta classifiers analyse and combine them, and send their own output to the

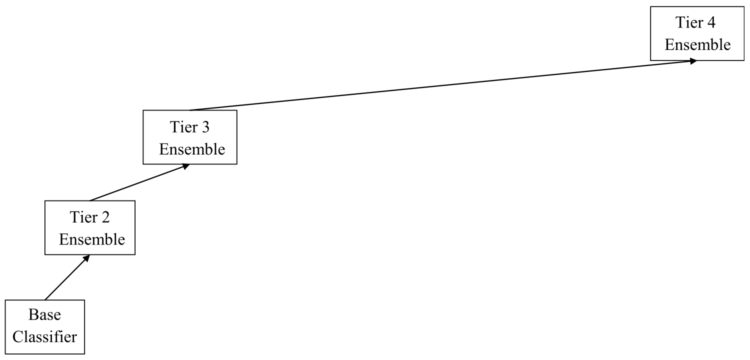


FIGURE 3. Initialization of a four-tier LIME classifier.

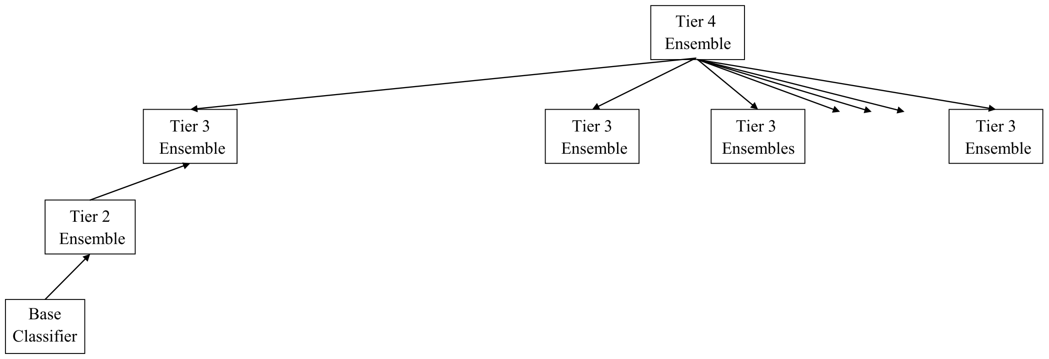
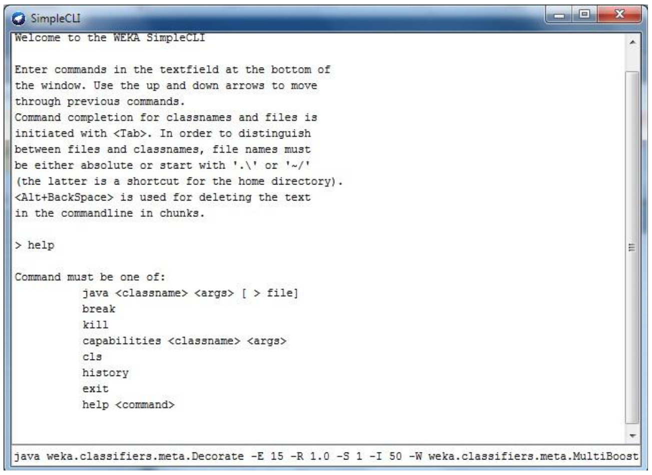


FIGURE 4. Stage 1 of generating four-tier LIME classifier. The fourth tier ensemble meta classifier generates third tier ensemble meta classifiers.



```
Welcome to the WEKA SimpleCLI

Enter commands in the textfield at the bottom of
the window. Use the up and down arrows to move
through previous commands.
Command completion for classnames and files is
initiated with <Tab>. In order to distinguish
between files and classnames, file names must
be either absolute or start with '.\' or '~/'
(the latter is a shortcut for the home directory).
<Alt+BackSpace> is used for deleting the text
in the commandline in chunks.

> help

Command must be one of:
          java <classname> <args> [ > file]
          break
          kill
          capabilities <classname> <args>
          cls
          history
          exit
          help <command>

java weka.classifiers.meta.Decorate -E 15 -R 1.0 -S 1 -I 50 -W weka.classifiers.meta.MultiBoost
```

FIGURE 5. A part of command line generating a four-tier LIME classifier in SimpleCLI.

fourth tier ensemble meta classifier. The fourth tier ensemble meta classifier analyses the results of the third tier ensemble meta classifiers and produces the final decision of the whole LIME classifier.

LIME classifiers belong to the general multitier and multistage approach considered, for example, in [22] and [23]. Many different multi-tier procedures and more general multi-classifier systems using base classifiers (but not ensemble meta classifiers) on several levels were explored previously, for example, in [22], [23], and [34]–[37] and have produced excellent results.

The new contribution of this article is in generating new large LIME systems as iterative ensembles of ensembles by linking a fourth tier ensemble meta classifier to another third tier ensemble meta classifier instead of a base classifier and linking the third tier ensemble meta classifier to a second tier ensemble meta classifier, which in turn are linked to their base classifiers. In this way the fourth tier ensemble meta classifier can generate the whole system. LIME classifiers are a new construction in the framework of this approach for the following two reasons. First, LIME classifiers include different ensemble meta classifiers on several tiers. Second, they use these ensemble meta classifiers iteratively to generate the whole classification system automatically.

This automatic generation capability includes many large ensemble meta classifiers in several tiers simultaneously and automatically combines them into one hierarchical unified system so that one ensemble meta classifier is an integral part of another one. This construction makes it easy to set up and run such large systems.

There are several advantages in using LIME construction. First, it generates the whole large system automatically, which makes it easy to set them up and run. This can enable

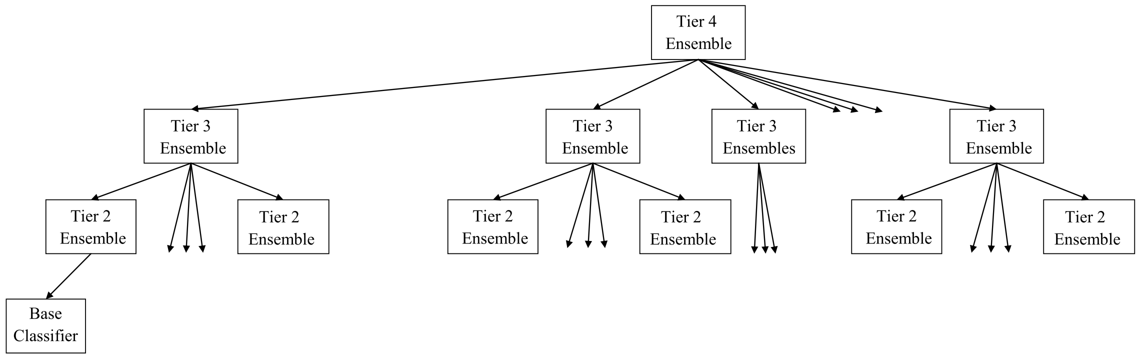


FIGURE 6. Stage 2 of generating four-tier LIME classifier. Each third tier ensemble meta classifier generates second tier ensemble meta classifiers.

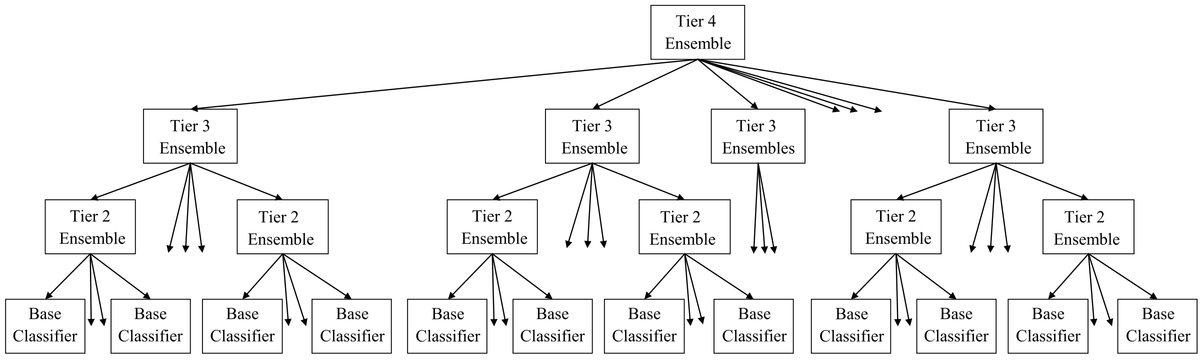


FIGURE 7. Stage 3 of generating four-tier LIME classifier. The direction of arrows indicates the generation of base classifiers by each second tier ensemble meta classifier.

practitioners to test many options before choosing the most appropriate one. Second, a LIME classifier can include different ensemble meta classifiers on several levels, combining the strengths of their methods.

The present article concentrates on the investigation of performance of LIME classifiers for the detection of malware. We carried out systematic experiments evaluating several various ensemble meta classifiers and their performance as parts of four tier LIME classifiers. The results presented here demonstrate that new four-tier LIME classifiers performed better than the base classifiers and standard ensemble meta classifiers included in the system. This example of application to the classification of malware shows that the new method of combining diverse ensemble meta classifiers into a unified hierarchical four-tier system can be applied to increase the performance of classifiers for handling Big Data. Every ensemble meta classifier at the third tier of this construction is an integral part of the ensemble meta classifier at the top tier. Likewise, every ensemble meta classifier of the second tier is a part of the ensemble meta classifier at the third tier. Finally, every base classifier at the bottom tier becomes a part of the ensemble meta classifier at the second tier. Iterative usage of one ensemble meta classifier as an integral part of another ensemble meta classifier makes it easy to set up, generate and run LIME classifiers.

The iterative method of automatic generation and training employed by the LIME classifiers has not been considered in the literature before. LIME classifiers can be implemented in the Weka SimpleCLI command line even though they can be very large. In our experiments, each four-tier LIME classifier contained 111 ensemble meta classifiers and 1000 base classifiers.

Large four-tier LIME classifiers require a lot of computer memory to be trained for Big Data, where they can be used to improve performance. If a data set is small, then the LIME classifier will revert to using just one base classifier and will produce the same outcomes as the base classifier. Experimental results in Section VII below demonstrate that such large four-tier LIME classifiers are effective if diverse ensembles are combined at different tiers of the four-tier LIME classifier. They are specifically designed to handle Big Data.

Theoretically, a LIME classifier generated in SimpleCLI as shown in Figure 5 can have any number of tiers. Each of its tiers can have as many classifiers (or ensemble meta classifiers) as required, since this number is defined by indicating just one command line parameter in SimpleCLI. We did not investigate n-tier LIME classifiers with $n > 4$, since such systems become very large and cannot be trained on our personal computer used for this study. We hope that a few years later personal computers will have sufficient memory for training LIME classifiers with larger numbers of tiers, and then the problem of investigating such *n*-tier LIME classifiers will be an interesting question for future work.

IV. BASE CLASSIFIERS

The following classifiers available in WEKA [38] were used as base classifiers in our experiments with outcomes presented in Section VII: BayesNet, DTNB, FURIA, J48, MultilayerPerceptron, NNge, Random Forest, SMO, and SPegasos. These robust classifiers were chosen since they represent most essential types of classifiers available in WEKA [38].

BayesNet implements a Bayesian network based on conditional probability distributions and a network structure.

DTNB creates a hybrid of decision table and naive Bayes classifiers using a forward selection search, where at each step selected attributes are modelled by a naive Bayes, the remainder by the decision table, and an option of dropping an attribute entirely from the model is also considered.

FURIA is a fuzzy unordered rule induction algorithm due to [39]. It extends the RIPPER algorithm by learning fuzzy rules instead of conventional rules and learning unordered rule sets instead of rule lists. At the same time it uses the same simple and comprehensive rule sets, and applies a novel rule stretching method. Experimental results show that FURIA outperforms RIPPER and J48, see [39].

J48 creates a C4.5 decision tree by adding attributes to the tree as explained in [40]. At every step the feature with the highest information gain is added. This means that every next attribute is chosen so that it is best in discriminating the instances in the training set. The classifier can generate pruned or unpruned C4.5 trees.

MultilayerPerceptron classifier trains a feedforward artificial neural network using backpropagation and sigmoid functions at each node.

NNge is based on a Nearest Neighbour relying on non-nested exemplars, which are hyperrectangles that can be viewed as if-then rules, as explained in [41] and [42].

Random Forest builds a forest of random trees by generating many decision tree predictors with randomly selected variable subsets and utilizing a different subset of training and validation data for each of these trees, as explained in [43]. To control the variation in generating the set of random trees, Random Forest uses the process of random selection of features proposed in [44]–[46]. After generating many trees, the resulting class prediction is based on votes from the trees. The variables are ranked and variables with lower rank are eliminated based on the basis of empirical performance heuristics [47]. The structure of random forest is represented in Figure 2.

SMO is a fast implementation of Support Vector Machines using Sequential Minimal Optimization. It generates a collection of hyperplanes in the *n*-dimensional space that separate classes of the data best and have large margins, i.e., distances to the nearest data points in the space, as explained in [48]–[50].

SPegasos performs the stochastic variant of the primal estimated sub-gradient solver for SVM (Pegasos) classifier.

We refer to [38] and [51] for more information on these base mining classifiers and their WEKA implementations.

V. ENSEMBLE META CLASSIFIERS

We used SimpleCLI command line in WEKA [38] to investigate the performance of the following ensemble meta classifier AdaBoost, Bagging, Dagging, Decorate, Grading, MultiBoost and Stacking.

AdaBoost uses several classifiers in succession. Each classifier is trained on the instances that have turned out more difficult for the preceding classifier. To this end all instances are assigned weights, and if an instance turns out difficult to classify, then its weight increases. We used the highly successful AdaBoost classifier described in [52].

Bagging (bootstrap aggregating), generates a collection of new sets by resampling the given training set at random and with replacement. These sets are called *bootstrap samples*. New classifiers are then trained, one for each of these new training sets. They are amalgamated via a majority vote, [53]., see also [54] and [55].

Dagging is useful in situations where the base classifiers are slow. It divides the training set into a collection of disjoint (and therefore smaller) stratified samples, trains copies of the same base classifier and averages their outputs using vote, [56].

Decorate involves constructing special artificial training examples to build diverse ensembles of classifiers. A comprehensive collection of tests have established that Decorate consistently creates ensembles more accurate than the base classifier, Bagging, Random Forests, which are also more accurate than Boosting on small training sets, and are comparable to Boosting on larger training sets, [57].

Grading trains base classifiers and grades their output as correct or wrong; these graded outcomes are then combined, [58].

MultiBoost extends the approach of AdaBoost with the wagging technique, [59]. Wagging is a variant of bagging where the weights of training instances generated during boosting are utilized in selection of the bootstrap samples, [60]. It is explained in [59] that experiments on a large and diverse collection of UCI data sets have demonstrated that MultiBoost achieves higher accuracy significantly more often than wagging or AdaBoost.

Stacking can be regarded as a generalization of voting, where meta-learner aggregates the outputs of several base classifiers, [61].

Let us refer to [38] and [51] for more information on these ensemble meta classifiers and their WEKA implementations.

VI. FEATURE EXTRACTION

The aim of this paper is to develop new LIME classifiers for analysis of Big Data. To facilitate the creation and preprocessing of the dataset, we do not introduce any new and sophisticated feature selection techniques and use simple and standard static features well known in the malware detection area. Generally speaking the collection of static features can be bypassed by obfuscation techniques. However, they remain widely used for malware detection and classification, since they can save processing time and can be applied when malware has only attempted to enter the system and has not been executed yet. For the purposes of our research additional changes to static features created by obfuscation can be regarded as an advantage, because considering new obfuscated versions of the same malware in a dataset as separate different instances of malware enriches the data and makes data more suitable for testing methods designed for Big Data.

In our experiments we used the byte sequences, or n-grams. They are sequences of n bytes read from an executable file to be classified. It is well known in the literature, that n-grams produce efficient static features for malware detection (cf. [62]–[64]). We applied the same method of feature extraction proposed in [65]. It uses TF-IDF scores to rank n-grams and reduce the number of sequences chosen as features.

The present article investigates a novel method for improving performance of the classifiers, and we did not attempt to extract more sophisticated collections of features. The extraction of features is very important for applications, for example, see [27] and [66], but it is not the main focus of the present article.

Since this paper concentrates on the contribution of four-tier LIME classifiers, for the purposes of this work, we extracted only a simple collection of the features. Our new experiments used simple features in a data set of malware from the industry partners of our laboratory, collected from the honeynet [67] and VH Heavens [68].

Following [65] and [69], we used *term frequency–inverse document frequency* sequence weights, or TF-IDF weights, to select n-grams in order to reduce the number of features. These weights are defined using the following concepts and notation (see [70] for more details).

Suppose that we are extracting features from a data set E, which consists of $|E|$ instances of malware and cleanware. For a sequence w and a instance of malware or cleanware m, let $N(w, m)$ be the number of times w occurs in m. Suppose that a collection $T = \{t_1, \ldots, t_k\}$ of terms $t_1, \ldots, t_k$ is being looked at. The *term frequency* of a word $w \in T$ in a instance of malware or cleanware m is denoted by $\mathrm{TF}(w, m)$ and is defined as the number of times w occurs in m, normalized over the number of occurrences of all terms in m:

$$\mathrm{TF}(w, m) = \frac{N(w, m)}{\sum_{i=1}^{k} N(t_i, m)} \tag{1}$$

The *document frequency* of the word w is denoted by $\mathrm{DF}(w)$ and is defined as the number of instances of malware and cleanware in the given data set where the sequence w occurs at least once. The *inverse document frequency* is used to measure the significance of each term. It is denoted by $\mathrm{IDF}(w)$ and is defined by the following formula

$$\mathrm{IDF}(w) = \log\left(\frac{|E|}{\mathrm{DF}(w)}\right). \tag{2}$$

The *term frequency–inverse document frequency* of a word w in instance of malware or cleanware m, or TF-IDF weight of w in m is defined by

$$\text{TF-IDF}(w, m) = \mathrm{TF}(w, m) \times \mathrm{IDF}(w, m). \tag{3}$$

VII. EXPERIMENTS EVALUATING PERFORMANCE

Our experiments are devoted to evaluating the performance of LIME classifiers for the detection of malware using big data.

It is critically important to conduct experiments and assess various classification schemes for processing of Big Data in particular areas. The outcomes of such experiments can be used to improve the performance of future practical implementations and can contribute to assessing further steps for future research. The performance of a classifier cannot be predicted on a purely theoretical basis. For any classification scheme that is able to produce very good outcomes in a specialised domain, there always exist other areas where different methods may turn out more effective. There are even theoretical results, known as ‘‘no-free-lunch’’ theorems, which imply that there does not exist a single algorithm that performs best for all problems [71].

We used 10-fold cross validation to evaluate the effectiveness of classifiers in all experiments. The following measures of performance of classifiers are often used in this research direction: precision, recall, F-measure, accuracy, sensitivity, specificity and Area Under Curve also known as the Receiver Operating Characteristic or ROC area.

Notice that weighted average values of the performance metrics are usually used. This means that they are calculated for each class separately, and a weighted average is found then. In particular, our results included in this paper deal with the weighted average values of precision. In contrast, the *accuracy* is defined for the whole classifier as the percentage of all instances classified correctly, which means that this definition does not involve weighted averages in the calculation. *Precision* of a classifier, for a given class, is the ratio of true positives to combined true and false positives.

Sensitivity is the proportion of positives (malware) that are identified correctly. *Specificity* is the proportion of negatives (legitimate software) which are identified correctly. Sensitivity and specificity are measures evaluating binary classifications. For multi-class classifications they can be also used with respect to one class and its complement. Sensitivity is also called True Positive Rate. *False Positive Rate* is equal to 1 - specificity. These measures are related to recall and precision. *Recall* is the ratio of true positives to the number of all positive samples (i.e., to the combined true positives and false negatives). The recall calculated for the class of malware is equal to sensitivity of the whole classifier.

All tables of outcomes in this paper include the Area Under Curve, AUC. For a given class, AUC is an area under the ROC graph that plots true positive rates for this class against false positive rates for a series of cut-off values. Equivalently, the ROC graph can be defined as a curve graphically displaying the trade-off between sensitivity and specificity for each cut-off value. The values of AUC belong to the range between 0.5 and 1, where 1 corresponds to perfect results, 0.5 is the worst possible value, and larger values of AUC correspond to better predictability of the classes.

In our experiments we used SimpleCLI in WEKA [38] to generate and execute all classifiers. AUC is a very well known measure of performance of classifiers and it is included in the standard output of all classifiers in WEKA produced in SimpleCLI.

First, we include the results of experiments comparing the performance of several base classifiers for malware. The results obtained for base classifiers are presented in Figure 8. Random Forest outperformed other base classifiers for the malware data set.

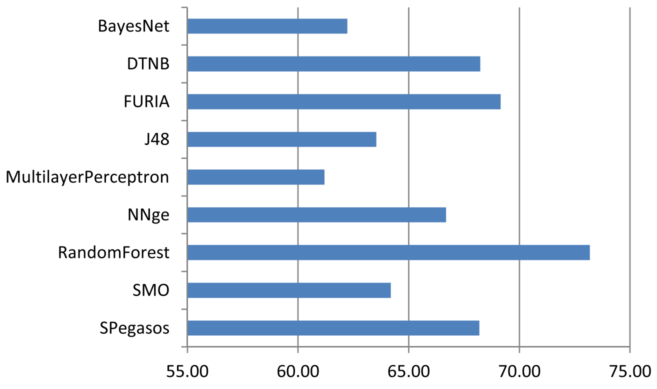


FIGURE 8. **AUC of base classifiers for security of big data.**

Second, we include the results of experiments comparing standard ensemble classifiers in their ability to improve the outcomes. We compared AdaBoost, Bagging, Dagging, Decorate, Grading, MultiBoost and Stacking based on Random Forest. AUCs of the resulting ensemble classifiers are presented in Figure 9, which shows improvement as compared to the base classifiers. In these tests all ensemble meta classifiers were used with one and the same base classifier, Random Forest, in all tests.

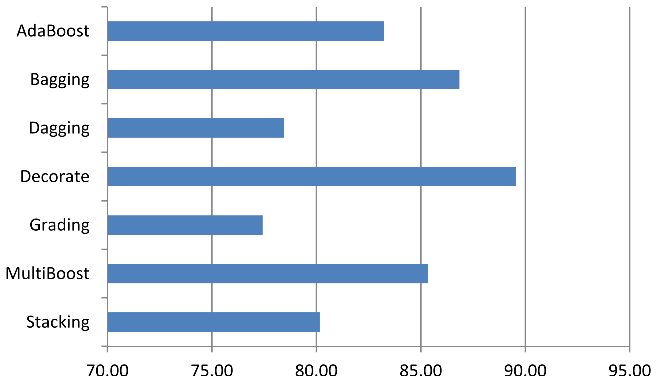


FIGURE 9. **AUC of ensemble classifiers for security of big data.**

Next, we include the results of experiments evaluating three-tier LIME classifiers. This is the main topic of the paper. These experiments included all combinations of Bagging, Decorate and MultiBoost, since these ensemble meta classifier produced better AUC in Figure 9. We have not included repetitions of the same ensemble meta classifier technique in both tiers, since tests have shown that such combinations do not produce improvement. The outcomes of the three-tier LIME classifiers are presented in Figure 10. A part of command generating one of these multi-level ensembles in SimpleCLI is shown in Figure 5.

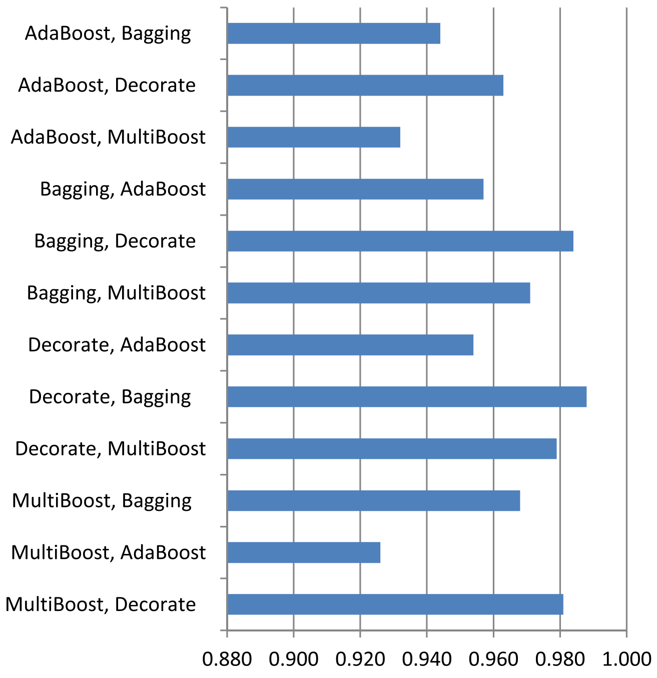


FIGURE 10. **AUC of three-tier LIME classifiers for security of big data.**

Finally, we include the results of experiments evaluating four-tier LIME classifiers. This is the main topic of the paper. These experiments included the four best combinations of three-tier LIME classifiers from Figure 10 improved using AdaBoost, Bagging, Decorate and MultiBoost at the top tier. The best outcomes of four-tier LIME classifiers are presented in Figure 11. The figure does not included repetitions of the same ensemble meta classifier technique in both tiers, since tests have shown that they do not produce further improvement.

VIII. DISCUSSION

Our work shows that large four-tier LIME classifiers are quite easy to use and can be applied to improve classifications, if diverse ensemble meta classifiers are combined at different tiers. It is an interesting question for future research to investigate LIME classifiers for other large datasets.

Random Forest outperformed other base classifiers for the malware data set, and Decorate improved its outcomes better than other ensemble meta classifiers did. The best outcome of AUC 0.998 was obtained by the four-tier LIME classifier

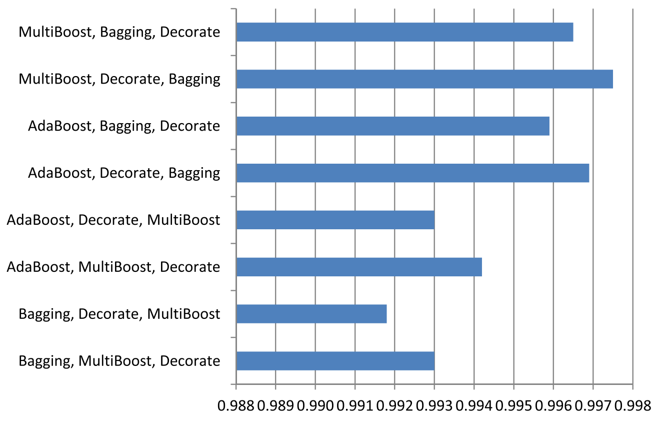


FIGURE 11. AUC of four-tier LIME classifiers for security of big data.

where MultiBoost was used at the fourth tier, Decorate was used at the third tier and Bagging was applied at the second tier.

The performance of ensemble meta classifiers considered in this paper depends on several numerical input parameters. In all experiments we used them with the same default values of these parameters in order to have a uniform equivalent comparison of outcomes across all of these ensemble meta classifiers.

IX. CONCLUSION

We introduced and investigated four-tier LIME classifiers originating as a contribution to the general approach considered by many authors. We obtain new results evaluating performance of such large four-tier LIME classifiers. These new results show, in particular, that Random Forest performed best in this setting, and that novel four-tier LIME classifiers can be used to achieve further improvement of the classification outcomes. The four-tier LIME classifiers based on Random Forest achieved better performance compared with the base classifiers or simpler ensemble meta classifiers. The four-tier LIME classifier with MultiBoost at the fourth tier, Decorate at the third tier and Bagging at the second tier obtained the best outcome with AUC 0.998.

We carried out a systematic investigation of new automatically generated four-tier LIME classifiers, where diverse ensemble meta classifiers are combined into a unified system by integrating different ensembles at the third and second tiers as parts of their parent ensemble meta classifiers at the higher tier. Our experiments evaluated the performance of these large four-tier LIME classifiers for a data set of malware and have demonstrated the feasibility and performance of the approach. The experimental outcomes show that four-tier LIME classifiers can be used to improve classifications. They are effective if diverse ensemble meta classifiers are combined at different tiers of the LIME classifier. They have made significant improvements to the performance of base classifiers and standard ensemble meta classifiers.

ACKNOWLEDGMENT

The authors are grateful to three reviewers for thorough reports with comments and corrections that have helped to improve this article.

REFERENCES

[1] J. Chen *et al.*, "Big data challenge: A data management perspective," *Frontiers Comput. Sci.*, vol. 7, pp. 157–164, Apr. 2013.
[2] L. Liu, "Computing infrastructure for big data processing," *Frontiers Comput. Sci.*, vol. 7, no. 2, pp. 165–170, 2013.
[3] X. Liu *et al.*, *The Design of Cloud Workflow Systems*. New York, NY, USA: Springer-Verlag, 2012.
[4] L. Wang, R. Ranjan, J. Chen, and B. Benatallah, *Cloud Computing: Methodology, System, and Applications*. Boca Raton, FL, USA: CRC Press, 2011.
[5] X. Liu, J. Chen, and Y. Yang, *Temporal QoS Management in Scientific Cloud Workflow Systems*. Amsterdam, The Netherlands: Elsevier, 2012.
[6] D. Yuan, Y. Yang, and J. Chen, *Computation and Storage in the Cloud: Understanding the Trade Offs*. Amsterdam, The Netherlands: Elsevier, 2013.
[7] C. Liu *et al.*, "An iterative hierarchical key exchange scheme for secure scheduling of big data applications in cloud computing," in *Proc. 12th IEEE Int. Conf. Trust Security Privacy Comput. Commun.*, Melbourne, Australia, Jul. 2013, pp. 9–16.
[8] H. Demirkan and D. Delen, "Leveraging the capabilities of service-oriented decision support systems: Putting analytics and big data in cloud," *Decision Support Syst.*, vol. 55, no. 1, pp. 412–421, 2013.
[9] L. Wang, J. Chen, and W. Jie, *Quantitative Quality of Service for Grid Computing: Applications for Heterogeneity, Large-Scale Distribution and Dynamic Environments*. Hershey, PA, USA: IGI Global, 2009.
[10] J. Chen and Y. Yang, *Effective and Efficient Temporal Verification in Grid Workflow—Enabling Timely Completion of Grid Workflow*. Saarbrücken, Germany: Lambert Academic Publishing, 2011.
[11] S. Rajan *et al.* (2013, Oct. 12). *Expanded Top Ten Big Data Security and Privacy Challenges*. Cloud Security Alliance, Los Angeles, CA, USA [Online]. Available: http://cloudsecurityalliance.org/research/big-data/
[12] M. D. Ryan, "Cloud computing security: The scientific challenge, and a survey of solutions," *J. Syst. Softw.*, vol. 86, no. 9, pp. 2263–2268, 2013.
[13] C. M. Rong, S. T. Nguyen, and M. G. Jaatun, "Beyond lightning: A survey on security challenges in cloud computing," *Comput. Electr. Eng.*, vol. 39, no. 1, pp. 47–54, 2013.
[14] L. Batten, J. Abawajy, and R. Dose, "Prevention of information harvesting in a cloud services environment," in *Proc. 1st Int. Conf. Cloud Comput. Services Science*, 2011, pp. 66–72.
[15] W. Dou, Q. Chen, and J. Chen, "A confidence-based filtering method for DDoS attack defense in cloud environment," *Future Generat. Comput. Syst.*, vol. 29, no. 7, pp. 1838–1850, 2013.
[16] S. Yu, S. Guo, and I. Stojmenovic, "Can we beat legitimate cyber behavior mimicking attacks from botnets?" in *Proc. 31st Annu. IEEE Int. Conf. Comput. Commun.*, Mar. 2012, pp. 2851–2855.
[17] X. Zhang, C. Liu, S. Nepal, C. Yang, and J. Chen, "Privacy preservation over big data in cloud systems," in *Security, Privacy and Trust in Cloud Systems*. Berlin, Germany: Springer-Verlag, 2013, pp. 239–257.
[18] Malware. (2013, Jul. 12). *Encyclopedia Britannica* [Online]. Available: http://www.britannica.com/EBchecked/topic/1477142/malware
[19] Malware. (2013, Jul. 12). *Wikipedia* [Online]. Available: http://en.wikipedia.org/wiki/Malware
[20] R. Islam, R. Tian, L. M. Batten, and S. Versteeg, "Classification of malware based on integrated static and dynamic features," *J. Netw. Comput. Appl.*, vol. 36, no. 2, pp. 646–656, 2013.
[21] I. R. A. Hamid and J. Abawajy, "Hybrid feature selection for phishing email detection," in *Algorithms and Architectures for Parallel Processing* (Lecture Notes in Computer Science), vol. 7017. Berlin, Germany: Springer-Verlag, 2011, pp. 266–275.
[22] R. Islam and J. Abawajy, "A multi-tier phishing detection and filtering approach," *J. Netw. Comput. Appl.*, vol. 36, no. 1, pp. 324–335, 2013.
[23] R. Islam, J. Abawajy, and M. Warren, "Multi-tier phishing email classification with an impact of classifier rescheduling," in *Proc. 10th ISPAN*, 2009, pp. 789–793.
[24] R. Islam, R. Tian, V. Moonsamy, and L. Batten, "A comparison of the classification of disparate malware collected in different time periods," *J. Netw.*, vol. 7, no. 6, pp. 956–955, 2012.

[25] V. Moonsamy, R. Tian, and L. Batten, "Feature reduction to speed up malware classification," in *Information Security Technology for Applications* (Lecture Notes in Computer Science), vol. 7161, P. Laud, Ed. Berlin, Germany: Springer-Verlag, 2012, pp. 176–188.
[26] W. Ma, P. Duan, S. Liu, G. Gu, and J.-C. Liu, "Shadow attacks: Automatically evading system-call-behavior based malware detection," *J. Comput. Virol.*, vol. 8, nos. 1–2, pp. 1–13, 2012.
[27] R. Islam, R. Tian, L. Batten, and S. Versteeg, "Classification of malware based on string and function feature selection," in *Proc. 2nd CTC Workshop*, 2010, pp. 9–17.
[28] W. Wang, I. Murynets, J. Bickford, C. Van Wart, and G. Xu, "What you see predicts what you get—Lightweight agent-based malware detection," *Security Commun. Netw.*, vol. 6, no. 1, pp. 33–48, 2013.
[29] R. Perdisci, D. Ariu, and G. Giacinto, "Scalable fine-grained behavioral clustering of HTTP-based malware," *Comput. Netw.*, vol. 57, no. 2, pp. 487–500, 2013.
[30] I. Santos, F. Brezo, X. Ugarte-Pedrero, and P. G. Bringas, "Opcode sequences as representation of executables for data-mining-based unknown malware detection," *Inf. Sci.*, vol. 231, pp. 64–82, May 2013.
[31] F. Shahzad, M. Shahzad, and M. Farooq, "In-execution dynamic malware analysis and detection by mining information in process control blocks of Linux OS," *Inf. Sci.*, vol. 231, pp. 45–63, May 2013.
[32] Z. Zhao, J. Wang, and C. Wang, "An unknown malware detection scheme based on the features of graph," *Security Commun. Netw.*, vol. 6, no. 2, pp. 239–246, 2013.
[33] M. Cimpoesu, D. Gavrilut, and A. Popescu, "The proactivity of perceptron derived algorithms in malware detection," *J. Comput. Virol.*, vol. 8, no. 4, pp. 133–140, 2012.
[34] R. Islam, J. Singh, A. Chonka, and W. Zhou, "Multi-classifier classification of spam email on a ubiquitous multi-core architecture," in *Proc. IFIP Int. Conf. NPC*, Oct. 2008, pp. 210–217.
[35] R. Islam and W. Zhou, "Email classification using multi-tier classification algorithms," in *Proc. 7th IEEE/ACIS Int. Conf. Comput. Inf. Sci.*, May 2008.
[36] R. Islam, W. Zhou, and M. U. Chowdhury, "Email categorization using (2+1)-tier classification algorithms," in *Proc. 7th IEEE/ACIS ICIS*, May 2008, pp. 276–281.
[37] R. Islam, W. Zhou, M. Gao, and Y. Xiang, "An innovative analyser for multi-classifier e-mail classification based on grey list analysis," *J. Netw. Comput. Appl.*, vol. 32, no. 2, pp. 357–366, 2009.
[38] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: An update," *SIGKDD Explorations Newslett.*, vol. 11, no. 1, pp. 10–18, 2009.
[39] J. C. Huehn and E. Huellermeier, "FURIA: An algorithm for unordered fuzzy rule induction," *Data Mining Knowl. Discovery*, vol. 19, no. 3, pp. 293–319, 2009.
[40] R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA, USA: Morgan Kaufmann, 1993.
[41] B. Martin, "Instance-based learning: Nearest neighbour with generalisation," Dept. Comput. Sci., Univ. Waikato, Hamilton, New Zealand, Tech. Rep. 95/18, 1995.
[42] S. Roy, "Nearest neighbor with generalization," M.S. thesis, Comput. Sci. Softw. Eng., Univ. Canterbury, Christchurch, New Zealand, 2002.
[43] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.
[44] T. K. Ho, "Random decision forest," in *Proc. 3rd Int. Conf. Document Anal. Recognit.*, 1995, pp. 278–282.
[45] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 8, pp. 832–844, Aug. 1998.
[46] Y. Amit and D. Geman, "Shape quantization and recognition with randomized trees," *Neural Comput.*, vol. 9, no. 7, pp. 1545–1588, 1997.
[47] L. Han, M. J. Embrechts, B. Szymanski, K. Sternickel, and A. Ross, "Random forests feature selection with K-PLS: Detecting ischemia from magnetocardiograms," in *Proc. ESANN*, vol. 14. 2006, pp. 221–226.
[48] T. Hastie and R. Tibshirani, "Classification by pairwise coupling," in *Advances in Neural Information Processing Systems*. Cambridge, MA, USA: MIT Press, 1998.
[49] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy, "Improvements to Platt's SMO algorithm for SVM classifier design," *Neural Comput.*, vol. 13, no. 3, pp. 637–649, 2001.
[50] J. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods*. Cambridge, MA, USA: MIT Press, 1998.
[51] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*. Amsterdam, The Netherlands: Elsevier, 2011.
[52] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *Proc. 13th Int. Conf. Mach. Learn.*, 1996, pp. 148–156.
[53] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, 1996.
[54] G. Liang, X. Zhu, and C. Zhang, "An empirical study of bagging predictors for imbalanced data with different levels of class distribution," in *Advances in Artificial Intelligence* (Lecture Notes in Artificial Intelligence), vol. 7106, D. Wang and M. Reynolds, Eds. Berlin, Germany: Springer-Verlag, 2011, pp. 213–222.
[55] Q. Sun and B. Pfahringer, "Bagging ensemble selection," in *Advances in Artificial Intelligence* (Lecture Notes in Artificial Intelligence), vol. 7106, D. Wang and M. Reynolds, Eds. Berlin, Germany: Springer-Verlag, 2011, pp. 251–260.
[56] K. M. Ting and I. H. Witten, "Stacking bagged and dagged models," in *Proc. 14th ICML*, 1997, pp. 367–375.
[57] P. Melville and R. J. Mooney, "Creating diversity in ensembles using artificial data," *Inf. Fusion*, vol. 6, no. 1, pp. 99–111, 2005.
[58] A. K. Seewald and J. Fuernkranz, "An evaluation of grading classifiers," in *Advances in Intelligent Data Analysis* (Lecture Notes in Computer Science), vol. 2189. Berlin, Germany: Springer-Verlag, 2001, pp. 115–124.
[59] G. I. Webb, "MultiBoosting: A technique for combining boosting and wagging," *Mach. Learn.*, vol. 40, no. 2, pp. 159–196, 2000.
[60] E. Bauer and R. Kohavi, "An empirical comparison of voting classification algorithms: Bagging, boosting, and variants," *Mach. Learn.*, vol. 36, nos. 1–2, pp. 105–139, 1999.
[61] D. H. Wolpert, "Stacked generalization," *Neural Netw.*, vol. 5, no. 2, pp. 241–259, 1992.
[62] M. Schultz, E. Eskin, E. Zadok, and S. Stolfo, "Data mining methods for detection of new malicious executables," in *Proc. IEEE Symp. Security Privacy*, May 2001, pp. 38–49.
[63] T. Abou-Assaleh, N. Cercone, V. Keselj, and R. Sweidan, "N-gram-based detection of new malicious code," in *Proc. 28th Annu. Int. Comput. Softw. Appl. Conf.*, Sep. 2004, pp. 41–42.
[64] J. Kolter and M. Maloof, "Learning to detect and classify malicious executables in the wild," *J. Mach. Learn. Res.*, vol. 7, pp. 2721–2744, Dec. 2006.
[65] A. Shabtai, R. Moskovitch, Y. Elovici, and C. Glezer, "Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey," *Inf. Security Tech. Rep.*, vol. 14, no. 1, pp. 16–29, 2009.
[66] X. Wang, W. Niu, G. Li, X. Yang, and Z. Shi, "Mining frequent agent action patterns for effective multi-agent-based web service composition," in *Agents and Data Mining Interation* (Lecture Notes in Artificial Intelligence), vol. 7103. Berlin, Germany: Springer-Verlag, 2012, pp. 211–227.
[67] C. Leita *et al.*, "The Leurre.com project: Collecting internet threats information using a worldwide distributed honeynet," in *Proc. WISTDCS*, Apr. 2008, pp. 40–57.
[68] (2012, Jun. 20). *VX Heavens Virus Collection* [Online]. Available: http://vx.netlux.org/vl.php
[69] G. Beliakov, J. Yearwood, and A. Kelarev, "Application of rank correlation, clustering and classification in information security," *J. Netw.*, vol. 7, no. 6, pp. 935–955, 2012.
[70] NIST/SEMATECH. (2013, Jul. 21). *E-Handbook of Statistical Methods* [Online]. Available: http://www.itl.nist.gov/div898/handbook/
[71] D. Wolpert, "The lack of a priori distinctions between learning algorithms," *Neural Comput.*, vol. 8, no. 7, pp. 1341–1390, 1996.

JEMAL H. ABAWAJY is a Professor and the Director of the Parallel and Distributed Computing Laboratory at Deakin University, Australia. He was an Organizing Committee Member of over 100 international conferences serving in various capacities, including Chair, General Co-Chair, Vice Chair, Best Paper Award Chair, Publication Chair, Session Chair, and Program Committee Member. He has authored more than 200 refereed articles, supervised numerous Ph.D. students to completion, and is on the editorial boards of many journals.

ANDREI KELAREV has authored two books and 194 journal articles, and is an Editor of five international journals. He was an Associate Professor with the University of Wisconsin and the University of Nebraska, USA, and a Senior Lecturer with the University of Tasmania, Australia. He was a Chief Investigator of a large Discovery grant from the Australian Research Council, and a Program Committee Member of several conferences. He is working for a research grant with the Parallel and Distributed Computing Laboratory, Deakin University, Australia.

MORSHED CHOWDHURY is a Faculty of the School of Information Technology and a Founder Member of the Parallel and Distributing Computing Laboratory at Deakin University, Australia. He has organized the IEEE sponsored ACIS2007 and SNPD2012 conferences as a Co-Chair. He was an Organizing Committee Member for over 50 international conferences serving in various capacities. He was a recipient of the Best Research Paper Award at the Annual International Conference on Information Technology Security in 2010. He has authored more than 70 refereed articles, and is an Editorial Board Member of the *International Journal of Software Innovation*, IGI Global, USA. He has been supervising seven Ph.D. students.