Peer-to-Peer Data Mining Classifiers for Decentralized Detection of Network Attacks

Walter Cerroni[2,3] **Gianluca Moro**[1,3] **Tommaso Pirini**[1] **Marco Ramilli**[2]

[1] Department of Computer Science and Engineering
[2] Department of Electrical, Electronic and Information Engineering "G. Marconi"
[3] CIRI - ICT, Information and Communication Technologies
University of Bologna,
Via Venezia 52, I-47521 Cesena (FC), Italy
Email: {walter.cerroni,gianluca.moro,tommaso.pirini,marco.ramilli}@unibo.it

Abstract

Data mining aims to extract from huge amount of data stochastic theories, called knowledge models, to explain or predict complex phenomenon. In this paper we propose new distributed data mining algorithms to recognize network attacks against a set of devices from statistic data generated locally by each device according to the standard Simple Network Management Protocol (SNMP) available in each modern operating systems. The idea is to place an autonomous mining resource in each network node that cooperates with its neighbors in a peer-to-peer fashion in order to reciprocally improve their detection capabilities. Differently from existing security solutions, which are based on centralized databases of attack signatures and transmissions of huge amount of raw traffic data, in this solution the network nodes exchange local knowledge models of few hundred bytes. The approach efficacy has been validated performing experiments with several types of attacks, with different network topologies and distributions of attacks so as to also test the node capability of detecting unknown attacks.

Keywords: Data Mining, Distributed Algorithms, AdaBoost, Network Attacks, Detection, Peer-to-Peer

1 Introduction

Data mining aims to automatically discover new knowledge from huge amount of data useful to explain or predict unknown phenomenon. Most of the current techniques have been developed for centralized systems where all the available data is collected in a single site. However, the growing need to apply these techniques to large data sets distributed over the network, has led to the deployment of distributed data mining algorithms (Klusch et al. 2003, da Silva et al. 2006, Lodi et al. 2009, 2010).

In this work we examine distributed data mining algorithms, in particular we focus on the development and evaluation of new classification algorithms for decentralized environments such as peer-to-peer systems (Monti & Moro 2008, 2009, Monti et al. 2010, Moro & Monti 2012). Recently several distributed data mining algorithms have been introduced, however the best approaches are inadequate to work in decentralized systems where nodes are in general autonomous and may arbitrarily leave and join the system, for instance because belong to different organizations or people.

We introduce two novel distributed classification algorithms taking inspiration from a well-known centralized algorithm called AdaBoost by Freund & Schapire (1995). AdaBoost has been used for distributed analysis and in parallel processing so far. The previous approaches deal with performance improvements aspects and data aggregation. In this work, the new AdaBoost algorithms are used to generate and share knowledge models across network of autonomous resources.

We then show how these algorithms have been applied to network security in which an attacker attacks a single or a group of host. In particular, we have investigated a collaborative behavior between network entities in which each one does not share huge amount of raw data, as it happens in decentralized systems, but rather sharing only knowledge models. The shared knowledge models, which consist only of few hundred bytes, are locally generated from local data according to the Simple Network Monitoring Protocol (SNMP), available in every operating systems. In our previous work, Cerroni et al. (2009), we achieved optimal results generating knowledge models according to an unsupervised solution, in which, differently from this new contribution, we shared SNMP data among peers.

The cooperation among peers benefits are mainly in the exchange of knowledge models. In the first place, this produces a strong decrease of the traffic amount in the network, for example with respect to the exchange of raw records. The latter would probably achieve the same result or better, but the records of which each peer derives its knowledge models with the mining algorithm can be huge and very dynamic - constantly changing. When the network have been distributed knowledge models generated by centralized data, these may no longer be valid because the environmental situation changes rapidly. This is true as far as the network is extensive and real. Exchanging the models the system exchange the same knowledge, because the mining algorithm is the same for each node, but in a higher level and more convenient. This is an advantage for the scalability of the system: if the network is very large, leading all records at each node or in a central node to be analyzed is not

This work was partially supported by the Italian MIUR Project *Autonomous Security* in the PRIN 2008 Programme.

convenient, also in terms of consistency, in modern dynamic scenarios. The knowledge models are useful to nodes that receive them because they represent guidelines to judge – in this example, classify – their raw data – in particular in this example data generated by the SNMP – according mining models derived from other data, that carriers new information. We want to reach the results obtained grouping into a single node all raw data for a centralized analysis and handing out the global knowledge models. This in many practical situations it is not possible for obvious reasons: dynamic environments, huge amount of raw data that changes quickly and continuously, very large networks, etc.

The paper is organized as follows: the related work 2 section describes the state-of-the-art in supervised mining algorithms; Section 3 introduces our two algorithms comparing them to the centralized version of AdaBoost algorithm and explaining the main difference between the two realized versions; Section 4 describes results and shows graphs about the attack network test case. Finally section 5 discusses conclusions and perspective works.

2 Related Work

Extracting information from very large distributed data-bases is a big challenge for data mining. Many databases are too big to be collected at a single site, and centralized training could be very slow in these specific cases. Today some databases are inherently distributed and cannot be unified for a long list of reasons.

For example, in a real-time environment, large amounts of data could be collected so often and in different patterns, which in the time required to make them consistent in a single structure would have lost their validity. As well, in distributed environments, it may happen that the effectiveness of certain information is limited to a single environment, so if you collected them all in a single database, some data may lose consistency.

Therefore it is very difficult to be able to design a single classification algorithm under these constraints. For example, the classification model construction to detect credit card frauds needs a huge strictly distributed data set. In addition, there are some databases having a large amounts of new data, available periodically or in real time. Re-training a new model on an entire data set it is both inefficient and expensive since each at time there is a significant increase or a change of data.

Some researchers have proposed the use of classification techniques to solve this problem. In general, standard algorithms have been developed, such as adding incremental induction on training of decision trees by Utgo (1994) and implementing an incremental ruled-based learner by Clearwater et al. (1989).

Another idea proposed is to parallelize learning algorithms - such as the *parallel* rule induction by Provost & Hennessy (1996), the construction of decision trees by Breiman & Spector (1994) and association rules by Agrawal et al. (1996), Han et al. (1997).

An alternative way is to combine multiple classifiers: this approach is also known as "the meta-learning".

Chan (1996) proposed to train classifiers on different training set partitions. Breiman Breiman (1999) used a statistical method to combine classifiers with voting, trained on a part of the original training set. The combination advantage is so the system can use many learning algorithms but every single algorithm is independent of the other one. Galtier et al. (2009) present a parallelization of AdaBoost by Freund & Schapire (1995) balancing workload on the "master-worker" strategy. Fan et al. (1999) studied two new techniques using AdaBoost to combine classifiers. In the first case, they only choose samples from the complete weighted training set to create classifiers are expected to be "weaker" than one trained from the complete training set. However, boosting can still increase the overall accuracy of the voted ensemble after many rounds. In the second case, they regard the AdaBoost weight updating formula as a way of assigning weights to classifiers in a weighted voting ensemble. With this approach, they have an opportunity to reuse old classifiers on new data sets and learn a new classifier to concentrate on portions of the new data where the old classifiers perform poorly.

These techniques use distributed and online learning techniques, and have been developed on JAM by Stolfo et al. (1997), a framework of data mining Agent-based systems. In general, the characteristics of software agents, such as autonomy, adaptability and decision-making, match very well to distributed system requirements, and also to distributed data mining.

3 Distributed AdaBoostM1 Algorithms

This section introduces the new algorithms Distributed AdaBoostM1 in multi-model and single-model versions, after a background illustration of the AdaBoostM1 algorithm by Freund & Schapire (1996).

3.1 AdaBoostM1

AdaBoostM1 is a widely used method, designed for classification. It is a meta-algorithm which uses different classification models according to a learning technique called *boosting* by Witten & Frank (2005).

Let assume that the learning algorithm is able to handle instances with a weight, represented by a positive number (we will review this assumption later). The weighted instances change the way it calculates the classifier error: in this case, error is the sum of the weights of misclassified instances, divided by the total weight of all the instances, instead of the fraction of misclassified instances. At each iteration, the learning algorithm focus on a particular set of instances, which has the highest weight. These records are important because there is a greater incentive to classify them properly. The C4.5 algorithm (Quinlan 1993, 1996, Kotsiantis 2007) is a learning method that can handle weighted instances.

The algorithm starts assigning the same weight to every instance of the training set, then calls the learning algorithm, which builds a classifier and assigns a new weight to each instance, based on the outcome of his analysis: the weight of instances correctly classified will be decreased and the weight of misclassified instances is increased. This produces a subset of "easy" instances, with low weight, and a subset of "hard" instances, with higher weight. In successive iterations, the generated classifiers focus their evaluation on "hard" instances and up to date the weights. It is possible to get different situations, for example, instances could become easier, or otherwise continuously increase their weight. After each iteration, the weights reflect how many times each instance has been misclassified by the classifiers produced up to that point. Maintaining a measure of the "difficulty" in each instance, this procedure provides an effective way to generate complementary classifiers.

How much weight should be changed after each iteration? It depends on the error of the overall classification. In particular, if e (a fraction between zero and one) denotes the error of a classifier with weighted data, then the weights of correctly classified instances will be updated as follows:

$$weight_{i+1} \leftarrow weight_i \times \frac{e}{1-e}$$

while misclassified instances will remain unchanged. This obviously does not increase the weight of misclassified instances, as stated earlier. However, after all weights have been updated, they are normalized so the weight of each misclassified instance increases and that of each correctly classified instance decreases.

Whenever the error on training data is weighted equal to or greater than 0.5, the current classifier is deleted and the algorithm stops. The same thing happens when the error is zero, because otherwise all the weights of the instances would be cancelled.

After the training session, we obtain a set of classifiers. In order to evaluate them, there is a voting system. A classifier that performs well on the training set (e close to zero) receives a high mark, while a classifier that performs bad (e close to 0.5) receives a low mark. In particular, the following equation is used for the assignment of votes:

$$vote = -\log \frac{e}{1-e},$$

which always returns a positive number. This formula also explains why the perfect classifiers on the training set must be eliminated: in fact, when e is equal to zero the weight of the vote is not defined. To classify a new instance you have to add the votes of all the classifiers and the class obtains the highest score is assigned to the instance.

At the beginning, we assumed that the learning algorithm is able to handle weighted instances. If not, however, it is possible to generate a set of unweighted instances by the weighted ones through resampling. Instead of changing the learning algorithm, it creates a new set replicating instances, proportional to their weight. As a result, high weight instances will be replicated frequently, while low weight instances could be not sampled. Once new set of data becomes large as the original, it replaces the method of learning.

A disadvantage of this procedure is given by the loss of information resulting from the repeal of some low weight instances from the data set. However, this can be turned into an advantage. When the learning algorithm generates a classifier whose error is greater than 0.5, the process of boosting must end if we use weighted data directly, but if you use a resampled data set, you could still produce a classifier with an error less of 0.5 generating a new resampled data set, maybe with a different *seed*. Resampling can be performed also when using the original version of the algorithm with weighted instances.

3.2 Distributed AdaBoostM1-MultiModel

In a distributed environment, mining algorithms, which are placed on every monitor node, create models of knowledge based on their training data. Exchanging models between neighbors they increase their knowledge.

During the first iteration of the algorithm, each monitor node runs the AdaBoostM1 algorithm on its training set. The result is a series of classification models - i.e., decision trees (assuming to use C4.5 mining algorithm). In this first phase, each mining engine does not care about their neighbors.

Algorithm 1 AdaBoostM1 pseudocode.

Require: $D_1(i) = 1/m,\ \forall\ i$ instances.
  for $t = 1 \rightarrow T$ **do**
    Call a *learner* using distribution D_t;
    Get back a classifier $c_t : X \rightarrow Y$;
    Calculate *error* of c_t, $e_t = \sum_{i:c_t(x_i) \neq y_i} D_t(i)$;
    if $e_t = 0 \wedge e_t > 0.5$ **then**
      $T = t - 1$;
    end if
    Set *vote* of c_t, $v_t = e_t/(1 - e_t)$;
    $D_{t+1} = \frac{D_t(i)}{N_t} \times \begin{cases} v_t & \text{if } c_t(x_i) = y_i \\ 1 & \text{otherwise;} \end{cases}$
    where N_t is a normalization constant.
  end for

Ensure: the *predicted class*:
  $\arg\ c_{fin}(x) = \arg\max_{y \in Y} \sum_{t:c_t(x_i)=y_i} \log(1/v_t)$

Once all nodes have generated models based on their local data, the algorithm continues with the next phase: knowledge sharing. Each node obtain the result of the previous iteration of every neighbors, the knowledge models previously built according to local data now are going to be changed according to the new neighbors data. In this way, in the global system, there is an exchange of information not in the form of data, but of classifiers, which offer an higher level of abstraction and less network traffic - since a model is smaller than a data set. In this context, however, we do not examine issues related to network communication between nodes, because the execution of the algorithm is simulated in a static way.

After that, classifiers collected from neighbors are added to classifiers generated on the local data to extend the knowledge of each monitor node. This knowledge is evaluated on a the test set of instances. Each test instance receives the class label that gets the most votes from all the available classifiers in the monitor node.

3.3 Distributed AdaBoostM1-SingleModel

This section describes a variant of the above algorithm which avoid the multiplication of the classification models on every monitor node. In scenarios where too much knowledge models are shared the Distributed AdaBoostM1-MultiModel algorithm might have issues such as: slowing down operations and a decreasing the accuracy of the results. The number of models in each node increases depending on the number of neighbors, causing a slowdown during the test to evaluate each record. The accuracy decreases as each model focuses on only a few attacks, received during the training of each node, and then those models do not know a particular attack issue a wrong result. When these models are the majority then the node can not detect the attack because good grades are the minority.

In order to fix the described issues we developed an algorithm which shares only a single model to every neighbor. The algorithm shares the best model rather than all the generated ones. The models generation remains unchanged, it follows the classical AdaBoostM1 algorithm. What changes is the sharing phase: when a node asks new models from its neighbors, the neighbors share only one model, the one who obtained the best score during the boosting phase. The monitor node itself, during the new in-

stance evaluation, does not consider all the models generated locally, but it chooses the best one among the local ones, and the best classifiers of its neighbors. The evaluation of new instances is done on a shorter list of models, and this reflects an improved efficiency in terms of transmission models to neighbors and during the evaluation of new instances.

Algorithm 2 Distributed algorithm pseudocode.

Require: m = maximum number of models generated by each node.
  for each node **do**
    Generate m knowledge models on local training set with *AdaBoostM1*.
    for each neighbor **do**
      Get m' neighbor models and add to its own knowledge base ($1 \leq m' \leq m$).
    end for
    Evaluate test set on its own knowledge models.
  end for

  for the Single-Model variant of the algorithm: $m' = 1$.

4 Experiments and Results

This section describes the experiments and the obtained results by applying the AdaBoostM1 to SNMP data gathered during attacks and normal network traffic. From fourteen SNMP parameters collected in a given time and during both attacks and normal network traffic, we want to be able to realize whether or not attacks are happening and possibly we want to distinguish between them.

We consider a scenario in which a set of monitoring stations, each of which collects raw data on network traffic, control a set of protected machines (i.e. servers, workstations...). The gathered information is provided through SNMP. Each monitor machine has a SNMP agent running to collect SNMP data on a protected machine.

The collected SNMP informations are organized into categories in a tree structure, known as MIB (Management Information Base): in this case, we consider the data related to TCP and IP network protocols, such as inbound and outbound packet counts. These MIBs represent the current state of the TCP/IP stack protected machine.

The idea is to analyze the given data through mining techniques to obtain classification models letting us understanding whether or not network attacks are happening. Since each of the monitoring stations collect data from a limited group of machines, it might be useful to gather such a data through a P2P network and use distributed data mining techniques: in this way each station can use the knowledge from its neighbors to expand and complete its own.

4.1 Experiment Setup

We considered 6758 observations as our data set, made with SNMP on a single machine. Each observation consists of the values of the 14 attributes listed in Table 1, defined by Cerroni et al. (2009). The collection of these observations has been divided into 6 sessions of different network traffic conditions, listed in Table 2. Data are classified according to the session during which they were collected, so that only one session corresponds to regular traffic and each other to different type of attack. These are well known attacks, recognized also by current IDS, so the results can be compared with them to evaluate the level of accuracy.

Table 1: Relevant Variables Considered from SNMP Data

Features Derived from SNMP Data
Number of processes in TCP listen state
Number of open TCP connections (any possible TCP state)
Number of TCP connections in time-wait state
Number of TCP connections in established state
Number of TCP connections in SYN-received state
Number of TCP connections in FIN-wait state
Number of different remote IP addresses with an open TCP connection
Remote IP address with the highest number of TCP connections
Remote IP address with the second highest number of TCP connections
Remote IP address with the third number of TCP connections
Local TCP port with the highest number of connections
Number of connections to the preceding TCP port
Local TCP port with the second highest number of connections
Number of TCP RST segments sent out

To obtain accurate information from the experiments, it was necessary to perform simulations with different parameters. In particular, we tried as many as possible combinations of network topologies, data distributions, and mining algorithms, to see the trend of results.

We considered these groups of parameters: those related to the algorithm, those related to the network topology, and those related to the data distribution. For each simulation, the main considered outcome was the accuracy of classification, calculated as the average percentage of test data, which are correctly classified by each node.

The algorithms used in the experiments are Distributed AdaBoostM1-MultiModel and DistributedAdaBoostM1-SingleModel. As stated above, they are the same mining algorithm (AdaBoostM1) that changes the exchanging knowledge with neighbors. In the first case, all the models generated locally are shared with each neighbor, in the last case, there is only one shared model for each monitor-node, that one got the lowest generalization error on the training set, that is the most accurate model.

Parameters of both algorithms are the following:

- as basic classifier to generate the models we used

Table 2: Simulated Traffic Sessions

Number	Description
0	Normal traffic
1	Denial of Service
2	Distributed Denial of Service
3	TCP Port Scanning using different techniques: FIN, SYN, ACK, WINDOW, NULL, XMAS
4	SSH Denial of Service
5	SSH Brute Force

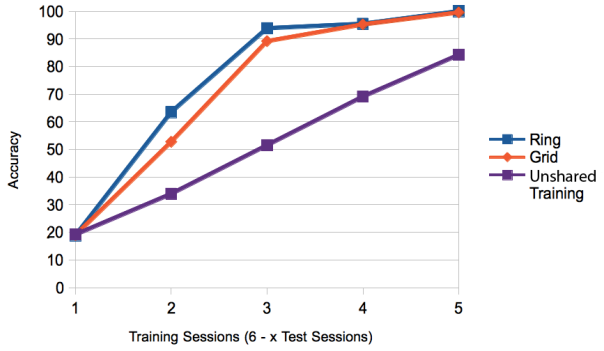

Figure 1: Accuracy of Distributed AdaBoostM1-MultiModel algorithm in a Ring Network and a Grid Network both of 64 nodes.

C4.5 (Quinlan 1993, 1996, Kotsiantis 2007), a decision tree algorithm. The confidence threshold for pruning was set at 0.3, and the minimum number of instances in the leaf of the trees was 2;

- the maximum number of models generated during boosting is 10;
- a single iteration was made for the exchange of models with the neighbors. This means that the shared knowledge is limited only to monitor-nodes that are just a "hop" from the current one. However, it's possible to increase the iterations of the algorithm to grow the knowledge base to second level neighbors and above. The results of these experiments will be studied in future work;
- the data distributor allocates to each node all instances of a number of randomly picked classes. Both training and test set are distributed according to this logic and independently from each other: so each node may have one or more classes in both training and test set.

The centralized case, bringing all of the training data set on a single site, with all types of attacks shown in Table 2, leads to an accuracy greater than 99%. This case, however, involves considerable cost in terms of network traffic, time and also requires a wide and general knowledge of the network, very difficult in modern systems.

We have analyzed instead a case where the training sets are distributed in the network. Each node is trained only on certain attacks, not all, and then tested on different attacks to show how the accuracy changes, betweens knowledge models sharing cases, according to the two algorithms presented in this work, and without models sharing. In the first case, the improvement is evident in all the graphics, since the case in which nodes are trained only on the half of the attack classes and tested on the other half. The unknown attacks are detected thanks to the neighbors models received.

4.2 Results

In this section we analyze some significant results that have been identified among the many performed simulations. In particular, the x-axis are arranged in order of increasing the number of training sessions and, in order of decreasing, the number of test sessions (6 - x). Training sessions as well as test ones are randomly selected from those available, with no repeats, but it's possible some sessions are in both sets, for each node.

The random number generator used to distribute the sessions at the nodes is controlled by a *seed* set as a parameter, which has been changed ten times, with values ranging from 100 to 109, for each simulation. Each value shown in the following graphs is therefore the average of ten simulations.

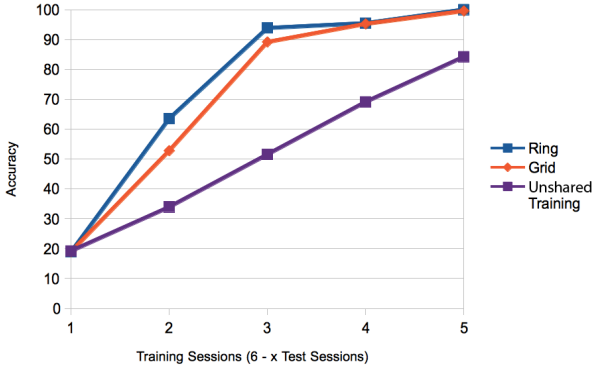

Figure 2: Accuracy of Distributed AdaBoostM1-SingleModel algorithm in a Ring Network and a Grid Network both of 64 nodes.

Figure 1 represents the trend of the accuracy of the Distributed AdaBoostM1-MultiModel algorithm on a ring network (with two neighbors per node) and a grid network (with four neighbors per node), as the number of training sessions. When the training sessions of each node grow, so the accuracy significantly improves. In particular, when the number of training sessions is three, half of those available, the accuracy is already around 90%. In this scenario, this means it's enough each node knows the half of the attacks to get a good accuracy with the help of neighbors, that is the exchange of knowledge models. However, a slightly better result is obtained in the ring network than in the grid network, so a higher number of neighbors can negatively influence the accuracy of nodes. This depends on the knowledge models shared by neighbors: in fact, some neighbors who don't know certain kinds of attacks provide models that do not work well. Hence, as a future work, it is necessary to find a way to evaluate the goodness of the models coming from the neighbors.

In Figure 2 there lines represents the same simulations but with the Distributed AdaBoostM1-SingleModel algorithm: the results are very similar to those of the previous algorithm, but this one have better performance, as less network traffic, because exchanges only one model instead of all, and as faster evaluation of new instances, because each node collects a number of models equal to the number of its neighbors plus one (the best model generated on local data set): $O(log\ n)$ instead of $O(m\ log\ n)$, where n is the maximum depth of decision trees and m is the number of models generated in boosting process in every node of network.

Figure 3 shows the Distributed AdaBoostM1-SingleModel algorithm dealing with six irregular networks, derived from the previous two regular topologies, namely the ring and the grid. The first three networks come from the same ring network of the previous simulations with the addition of 10%, 15% and 20% of random links between nodes. Added random links are created between nodes previously not connected and to increase the degree of network connectivity as the indicated percentage. The result shows

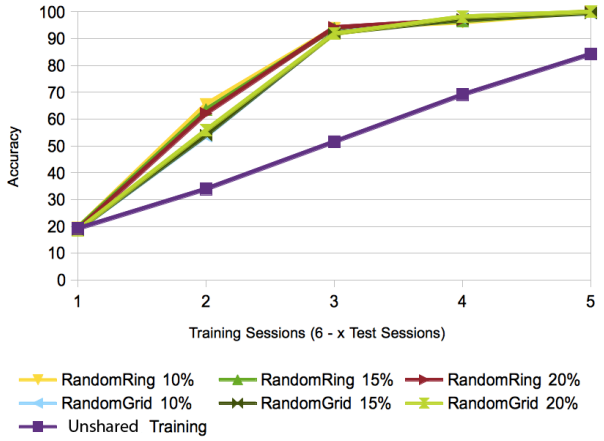

Figure 3: Accuracy of Distributed AdaBoostM1-SingleModel algorithm in two types of regular networks (Ring and Grid, with 64 nodes) adding in each one different percentages of random links, respectively 10%, 15% and 20%.

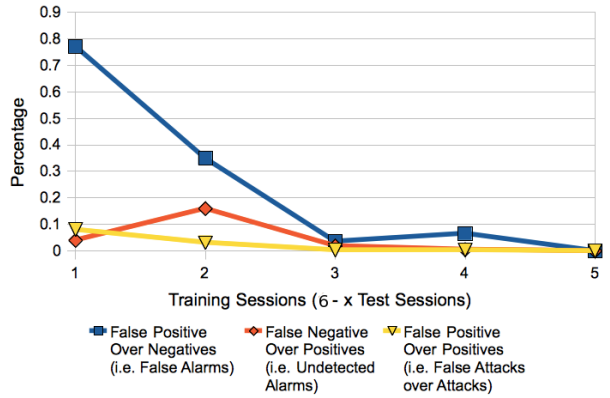

Figure 4: False Positive Rates between Normal Traffic and Attacks with Distributed AdaBoostM1-SingleModel algorithm in a Ring Network of 64 nodes.

the increase of links don't significantly affect the accuracy of the recognition of the attacks.

Finally, Figure 4 and Figure 5 show the false positive rates derived from the simulations of Figure 2 with the Distributed AdaBoostM1-SingleModel algorithm, respectively for the ring network and the grid network. The line of points identified with squares represents the percentage of false alarms, that is normal traffic recognized as attack by the system. When training sessions are few (one or two) there are quite high peaks, especially in the grid network, the peak is close to 4%, due to the fact that many nodes may have never seen the session of the normal traffic, because sessions are distributed casually by the data distributor, and therefore may not be able to recognize it, with the help of neighbors too. On the contrary, as shown by the line of diamonds, the unidentified attacks always remain around 0.1%. The last line of points, identified by triangles, represents the false attacks compared to attacks detected. If it were high it would means the system is not able to detect enough attacks, therefore is bad. In this case, this line is usually below 0.1%, so false alarms detected by the system are a very small part compared to all correctly detected attack.

Figure 6 shows two samples of decision trees generated by the algorithm C4.5

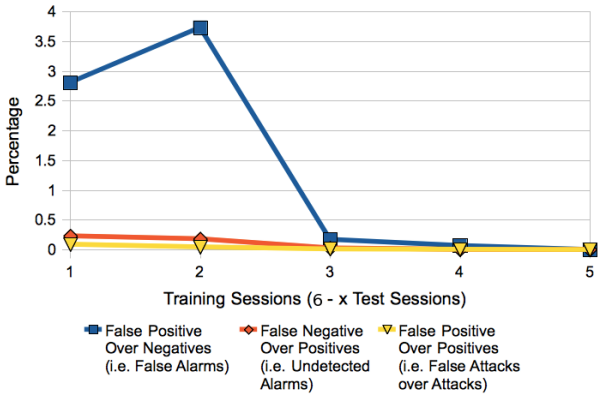

Figure 5: False Positive Rates between Normal Traffic and Attacks with Distributed AdaBoostM1-SingleModel algorithm in a Grid Network of 64 nodes.

(`weka.classifiers.trees.J48`) implemented in WEKA with the same parameters used in the simulations. The first and largest tree is generated with all the six attack classes, the smaller one instead is generated with three attack classes, in particular "0","2" and "5". We calculate the amount of network traffic generated by our distributed algorithms in the worst case. We consider the size of the decision tree generated by all classes of attack: about 1 Kbyte. The average size of the models however is much smaller because in general it is generated by a smaller number of classes. In the grid network there are 64 nodes, each node has 4 neighbors, and the Distributed AdaBoostM1-MultiModel algorithm shares up to 10 models (decision trees), so the total amount of bytes flowing in our system, in worst case, is:

$$10 \times 4 \times 64 = 2560 Kbyte$$

We consider also a centralized situation, where all the data are reached by a single node to analyze them and after distribute the generated model to the entire network. We consider the best case to highlight the benefits of our approach. Each node has accordingly the data related to only one attack class: about 1100 of the 6758 data set observations, corresponding in general to 100 Kbytes. To reach a node from any point of a ring network (simpler than a grid network), the data takes a average of $N/4$ hops ($64/4 = 16$). The network has 64 nodes as above, so:

$$100 \times 16 \times 64 = 102400 Kbyte$$

At this number, which is already two orders of magnitude larger than our worst case, we must add the traffic produced by the distribution of the generated knowledge model (1 Kbyte) to every node of the network.

$$1 \times 16 \times 64 = 1024 Kbyte$$

This simple estimate shows the benefit of our work in network traffic, compared to a centralized solution.

5 Conclusions and Future Works

We introduced two distributed data mining algorithms called Distributed AdaBoosM1-MultiModel and Distributed AdaBoostM1-SingleModel. Both algorithms and the underlaying framework for the generation of several networks and attack scenarios

```
Scheme:       weka.classifiers.trees.J48 -C 0.3 -M 2
Instances:    6758
Attributes:   16

J48 pruned tree
------------------

C_tcpConnectionState_established <= 2
|   C_tcpConnectionState_synReceived <= 2
|   |   tcpOutRsts <= 1: 0 (591.66)
|   |   tcpOutRsts > 1
|   |   |   C_tcpConnectionState_established <= 0
|   |   |   |   C_tcpListenerProcess <= 0: 0 (8.34)
|   |   |   |   C_tcpListenerProcess > 0: 3 (12.0)
|   |   |   C_tcpConnectionState_established > 0: 3 (1040.0)
|   C_tcpConnectionState_synReceived > 2
|   |   2IP_Connections <= 2
|   |   |   1IP_Connections <= 35
|   |   |   |   C_tcpConnectionState <= 110: 1 (24.0)
|   |   |   |   C_tcpConnectionState > 110: 2 (3.0)
|   |   |   1IP_Connections > 35: 1 (1685.0)
|   |   2IP_Connections > 2: 2 (1380.0)
C_tcpConnectionState_established > 2
|   C_tcpConnectionState_timeWait <= 0: 4 (1003.0)
|   C_tcpConnectionState_timeWait > 0: 5 (1011.0)

Number of Leaves  :     10

Size of the tree :      804 byte

Time taken to build model: 0.37 seconds


#############################################################


Scheme:       weka.classifiers.trees.J48 -C 0.3 -M 2
Instances:    2994
Attributes:   16

J48 pruned tree
------------------

C_tcpConnectionState_established <= 2
|   C_tcpConnectionState_synReceived <= 0: 0 (600.0)
|   C_tcpConnectionState_synReceived > 0: 2 (1383.0)
C_tcpConnectionState_established > 2: 5 (1011.0)

Number of Leaves  :     3

Size of the tree :      197 byte

Time taken to build model: 0.26 seconds
```

Figure 6: Decision tree samples generated by WEKA using C4.5 algorithm (`weka.classifiers.trees.J48`).

have been fully implemented as new components of WEKA[1].

Both algorithms work in purely decentralized scenarios where nodes exchange only local knowledge models of few bytes rather than huge amount of network traffic as performed by most of existing solutions. In particular each network node extracts and shares knowledge models from local SNMP data.

The models produced by a node, against a certain type of attack become useful for another node not previously aware of that attack. This allows nodes to be able to recognize unknown attacks and even to prevent them. The knowledge sharing allow each node to build a more complete knowledge base, compared to that produced using only their local data.

The experimental results show that both algorithms can provide accurate classifications, even in case of unknown attacks, moreover the best performance are not far from the ideal solution where all data are centralized in a single machine, which for this reason cannot scale as the network dimension increase.

The algorithms can be extended in several directions that are beyond the scope of this work, such as dealing also with malicious nodes interested in exchanging bad knowledge to reduce the global accuracy or improving the knowledge in order to identify bot-nets as well.

[1]Data Mining open source tool developed by the University of Waikato (NZ); http://www.cs.waikato.ac.nz/ml/weka/

References

Agrawal, R., Mehta, M., Shafer, J., Srikant, R., Arning, A. & Bollinger, T. (1996), The Quest data mining system, *in* 'Proceedings of the 2nd International Conference on Knowledge Discovery in Databases and Data Mining', pp. 244–249.

Breiman, L. (1999), 'Pasting small votes for classification in large databases and on-line', *Machine Learning* **36**(1), 85–103.

Breiman, L. & Spector, P. (1994), Parallelizing CART using a workstation network, *in* 'Proc Annual American Statistical Association Meeting'.

Cerroni, W., Monti, G., Moro, G. & Ramilli, M. (2009), 'Network Attack Detection Based on Peer-to-Peer Clustering of SNMP Data', *Quality of Service in Heterogeneous Networks* pp. 417–430.

Chan, P. (1996), 'An extensible meta-learning approach for scalable and accurate inductive learning'.

Clearwater, S., Cheng, T., Hirsh, H. & Buchanan, B. (1989), Incremental batch learning, *in* 'Proceedings of the sixth international workshop on Machine learning', Morgan Kaufmann Publishers Inc., pp. 366–370.

da Silva, J. C., Klusch, M., Lodi, S. & Moro, G. (2006), 'Privacy-preserving agent-based distributed data clustering', *Web Intelligence and Agent Systems* **4**(2), 221–238. `http://iospress.metapress.com/content/1dva3ew1f2emam44/`.

Fan, W., Stolfo, S. & Zhang, J. (1999), The application of AdaBoost for distributed, scalable and on-line learning, *in* 'Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining', ACM, pp. 362–366.

Freund, Y. & Schapire, R. (1995), A desicion-theoretic generalization of on-line learning and an application to boosting, *in* 'Computational learning theory', Springer, pp. 23–37.

Freund, Y. & Schapire, R. (1996), Experiments with a new boosting algorithm, *in* 'Machine Learning-International Workshop Then Conference', Citeseer, pp. 148–156.

Galtier, V., Genaud, S. & Vialle, S. (2009), Implementation of the AdaBoost Algorithm for Large Scale Distributed Environments: Comparing JavaSpace and MPJ, *in* '2009 15th International Conference on Parallel and Distributed Systems', IEEE, pp. 655–662.

Han, E., Karypis, G. & Kumar, V. (1997), Scalable parallel data mining for association rules, *in* 'Proceedings of the 1997 ACM SIGMOD international conference on Management of data', ACM, pp. 277–288.

Klusch, M., Lodi, S. & Moro, G. (2003), Distributed clustering based on sampling local density estimates, *in* G. Gottlob & T. Walsh, eds, 'IJCAI', Morgan Kaufmann, pp. 485–490. `http://www-ags.dfki.uni-sb.de/~klusch/papers/ijcai03-KDEC-paper.pdf`.

Kotsiantis, S. (2007), 'Supervised machine learning: A review of classification techniques', *Emerging artificial intelligence applications in computer engineering: real word AI systems with applications in eHealth, HCI, information retrieval and pervasive technologies* p. 3.

Lodi, S., Monti, G., Moro, G. & Sartori, C. (2009), Peer-to-peer data clustering in self-organizing sensor networks, *in* 'Intelligent Techniques for Warehousing and Mining Sensor Network Data', IGI Global, Information Science Reference, December 2009, Hershey, PA, USA, pp. 179–211. http://www.igi-global.com/chapter/peer-peer-data-clustering-self/39546.

Lodi, S., Moro, G. & Sartori, C. (2010), Distributed data clustering in multi-dimensional peer-to-peer networks, *in* 'Proceedings of the Twenty-First Australasian Conference on Database Technologies - Volume 104', ADC '10, Australian Computer Society, Inc., Darlinghurst, Australia, Australia, pp. 171–178. http://dl.acm.org/citation.cfm?id=1862242.1862264.

Monti, G. & Moro, G. (2008), Multidimensional range query and load balancing in wireless ad hoc and sensor networks, *in* 'Eighth IEEE International Conference on Peer-to-Peer Computing, 2008', IEEE Computer Society, pp. 205–214. http://doi.ieeecomputersociety.org/10.1109/P2P.2008.27.

Monti, G. & Moro, G. (2009), Self-organization and local learning methods for improving the applicability and efficiency of data-centric sensor networks, *in* 'Quality of Service in Heterogeneous Networks (QSHINE 2009)', Vol. 22 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, Springer Berlin Heidelberg, pp. 627–643. http://dx.doi.org/10.1007/11942634_40.

Monti, G., Moro, G., Tufano, G. & Rosetti, M. (2010), Self-organizing mobile mesh networks with peer-to-peer routing and information search services, *in* 'Proceedings of The Third International Conference on Advances in Mesh Networks (MESH 2010)', IEEE Computer Society, Venezia, Italy, pp. 17–22. http://doi.ieeecomputersociety.org/10.1109/MESH.2010.14.

Moro, G. & Monti, G. (2012), 'W-grid: A scalable and efficient self-organizing infrastructure for multi-dimensional data management, querying and routing in wireless data-centric sensor networks', *Journal of Network and Computer Applications* **35**(4), 1218–1234. http://dx.doi.org/10.1016/j.jnca.2011.05.002.

Provost, F. & Hennessy, D. (1996), Scaling up: Distributed machine learning with cooperation, *in* 'Proceedings of the National Conference on Artificial Intelligence', Citeseer, pp. 74–79.

Quinlan, J. (1993), *C4. 5: programs for machine learning*, Morgan Kaufmann.

Quinlan, J. (1996), 'Improved use of continuous attributes in C4. 5', *Arxiv preprint cs/9603103* .

Stolfo, S., Prodromidis, A., Tselepis, S., Lee, W., Fan, D. & Chan, P. (1997), JAM: Java agents for meta-learning over distributed databases, *in* 'Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining', pp. 74–81.

Utgo, P. (1994), An improved algorithm for incremental induction of decision trees, *in* 'Proceedings of the Eleventh International Conference on Machine Learning', Citeseer, pp. 318–325.

Witten, I. & Frank, E. (2005), *Data Mining: Practical machine learning tools and techniques*, Morgan Kaufmann Pub.