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

Walter Cerroni\textsuperscript{2,3}, Gianluca Moro\textsuperscript{1,3}, Tommaso Pirini\textsuperscript{1} and Marco Ramilli\textsuperscript{2}

\textsuperscript{1} Department of Computer Science and Engineering
\textsuperscript{2} Department of Electrical, Electronic and Information Engineering “G. Marconi”
\textsuperscript{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, Lodì 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 system. 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
The combination advantage is so the system can use voting, trained on a part of the original training set. Breiman (1999) learning.

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 (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).

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.

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:

\[
\text{weight}_{t+1} = \text{weight}_t \times \frac{e}{1-e}
\]

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.

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 a higher level of abstraction and less network traffic - since a model is smaller than a data set. 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 (\( 1 \leq m' \leq m \)).
    Evaluate test set on its own knowledge models.
  end for
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-node 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 collects 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.

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 Distributed AdaBoostM1-SingleModel. As stated above, they are the same mining algorithm (AdaBoostM1) that changes the exchanging knowledge with neighbors. 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 1: Relevant Variables Considered from SNMP Data**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of processes in TCP listen state</td>
<td></td>
</tr>
<tr>
<td>Number of open TCP connections (any possible TCP state)</td>
<td></td>
</tr>
<tr>
<td>Number of TCP connections in time-wait state</td>
<td></td>
</tr>
<tr>
<td>Number of TCP connections in established state</td>
<td></td>
</tr>
<tr>
<td>Number of TCP connections in SYN-received state</td>
<td></td>
</tr>
<tr>
<td>Number of TCP connections in FIN-wait state</td>
<td></td>
</tr>
<tr>
<td>Number of different remote IP addresses with an open TCP connection</td>
<td></td>
</tr>
<tr>
<td>Remote IP address with the highest number of TCP connections</td>
<td></td>
</tr>
<tr>
<td>Remote IP address with the second highest number of TCP connections</td>
<td></td>
</tr>
<tr>
<td>Remote IP address with the third number of TCP connections</td>
<td></td>
</tr>
<tr>
<td>Local TCP port with the highest number of connections</td>
<td></td>
</tr>
<tr>
<td>Number of connections to the preceding TCP port</td>
<td></td>
</tr>
<tr>
<td>Local TCP port with the second highest number of connections</td>
<td></td>
</tr>
<tr>
<td>Number of TCP RST segments sent out</td>
<td></td>
</tr>
</tbody>
</table>

**Table 2: Simulated Traffic Sessions**

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal traffic</td>
</tr>
<tr>
<td>1</td>
<td>Denial of Service</td>
</tr>
<tr>
<td>2</td>
<td>Distributed Denial of Service</td>
</tr>
<tr>
<td>3</td>
<td>TCP Port Scanning using different techniques: FIN, SYN, ACK, WINDOW, NULL, XMAS</td>
</tr>
<tr>
<td>4</td>
<td>SSH Denial of Service</td>
</tr>
<tr>
<td>5</td>
<td>SSH Brute Force</td>
</tr>
</tbody>
</table>
Figure 1: Accuracy of Distributed AdaBoostM1-MultiModel algorithm in a Ring Network and a Grid Network both of 64 nodes.

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

Figure 3: Accuracy of 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, 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 5: False Positive Rates between Normal Traffic and Attacks with Distributed AdaBoostM1-SingleModel algorithm in a Grid Network of 64 nodes.

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

$$10 \times 4 \times 64 = 2560 \text{Kbyte}$$

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:

$$100 \times 16 \times 64 = 102400 \text{Kbyte}$$

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.

$$1 \times 16 \times 64 = 1024 \text{Kbyte}$$

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 AdaBoostM1-MultiModel and Distributed AdaBoostM1-SingleModel. Both algorithms and the underlaying framework for the generation of several networks and attack scenarios...
Figure 6: Decision tree samples generated by WEKA using C4.5 algorithm (weka.classifiers.trees.J48).

References


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