Going a Step Beyond the Black and White Lists for URL Accesses in the Enterprise by Means of Categorical Classifiers

A. M. Mora, P. De las Cuevas and J. J. Merelo
Depto. Arquitectura y Tecnologa de Computadores, ETSIIT-CITIC, University of Granada, Granada, Spain

Keywords: Data Mining, Corporate Security Policies, URL request, Machine Learning, Classification.

Abstract: Corporate systems can be secured using an enormous quantity of methods, and the implementation of Black or White lists is among them. With these lists it is possible to restrict (or to allow) the users the execution of applications or the access to certain URLs, among others. This paper is focused on the latter option. It describes the whole processing of a set of data composed by URL sessions performed by the employees of a company; from the preprocessing stage, including labelling and data balancing processes, to the application of several classification algorithms. The aim is to define a method for automatically make a decision of allowing or denying future URL requests, considering a set of corporate security policies. Thus, this work goes a step beyond the usual black and white lists, since they can only control those URLs that are specifically included in them, but not by making decisions based in similarity (through classification techniques), or even in other variables of the session, as it is proposed here. The results show a set of classification methods which get very good classification percentages (95-97%), and which infer some useful rules based in additional features (rather that just the URL string) related to the user's access. This led us to consider that this kind of tool would be very useful tool for an enterprise.

1 INTRODUCTION

With the diffusion and evolution in the society of the so-called *smartphones*, a new scenario defined by the Bring Your Own Device (BYOD) tendency has been created, meaning that people do not use their smart devices for only one purpose (personal life or business) anymore. This scenario, in which the devices that access to the company system are owned by the users (employees), and that could contain both personal and professional information, has turned the security focus on those users, who have become one of the main threats (even not on purpose) to the corporate security (Oppliger, 2011).

This has meant the rising of new security issues, which are normally dealt by means of *Corporate Security Policies* (CSPs), which are basically a set of security rules aimed to protect the company assets, by defining permissions to be considered for every different action to be performed inside the security system.

These CSPs usually include policies that allow or deny access to non-confident (or non-certified) web sites (referenced by their URLs in this work). Moreover, several web pages might be also controlled for productivity or suitability reasons. Thus, some of the CSPs usually define sets of allowed or denied pages/sites that could be accessed by the enterprise users/employees. These sets are usually included in a White (permitted) or Black (non-permitted) Lists. These lists act as a good control tool for those URLs included in them as well as for the complementary, i.e. the URLs not included in a Whitelist have automatically denial of access, for instance.

In this work we go a step beyond, trying to define a tool for automatically making an allowance or denial decision with respect to URLs that are not included in the aforementioned lists. This decision would be based in that one made for similar URL accesses (those with similar features), but considering other parameters of the request/connection instead of just the URL string, as those lists do.

Thus, the problem has been transformed into a *classification* one, in which we have started from a set of unlabelled patterns, that model the connection properties from a huge amount of *real*[1] URL accesses (known as sessions). Then we have assigned a label to many of them, considering a set of *real*[2] security

[1] Taken from a log file given by a volunteer Spanish company.

[2] The set of rules has been written by the same company, with respect to its employees.

Mora A., Cuevas P. and Merelo J..
Going a Step Beyond the Black and White Lists for URL Accesses in the Enterprise by Means of Categorical Classifiers.
DOI: 10.5220/0005170601250134
In *Proceedings of the International Conference on Evolutionary Computation Theory and Applications* (ECTA-2014), pages 125-134
ISBN: 978-989-758-052-9

rules (CSPs) defined by the Chief Security Officer (CSO) in the company. The resulting dataset has been processed by means of different classification methods, in order to find the best algorithm for dealing with these data. Previously, data balancing techniques were applied, namely *undersampling* and *oversampling* (Japkowicz and Stephen, 2002), due to the high imbalance present in the dataset (more than two thirds of the patterns belonged to the majority class).

Different data partitions have been done in the experiments, even considering consecutive URL sessions in the training and test files. The results are quite good, getting classification accuracies around 95-97% in the test phase, even when using the unbalanced datasets. Then, after analysing the yielded sets of classification rules and trees, several rules can be identified, based in other features rather than the URL itself, which is the aim of this work.

The paper is structured as follows. Next section describes related work in relation to the application of Data Mining and Machine Learning techniques to security issues inside a company. Section 3 presents the problem we solve and the dataset we have worked with. The followed methodology is described in Section 4, concerning the data preprocessing and a first round of experiments comparing different classification methods. Once the best of them were selected, a set of experiments have been conducted, and the results are described and discussed in Section 5. Finally, the conclusions and future lines of research are presented in Section 6.

2 STATE OF THE ART

Our work tries to obtain a URL classification tool for enhancing the security in the client side, as at the end we want to get if a certain URL is secure or not, having as reference a set of rules (derived from a CSP) that allow or deny a set of known *http* requests. For this, Data Mining (DM) and Machine Learning (ML) techniques have been applied. This section gives an overview in a number of solutions given to protect the user, or the company, against unsecure situations.

Due to the nature of the data (URL accesses performed by humans), the used set of data is highly unbalanced (Chawla, 2005). In order to deal with this problem there exist several methods in the literature, but all of them are mainly grouped in three techniques (Japkowicz and Stephen, 2002):

- *Undersampling the Over-sized Classes:* i.e. reduce the considered number of patterns for the classes with the majority.
- *Oversampling the Small Classes:* i.e. introduce additional (normally synthetic) patterns in the classes with the minority.
- *Modifying the Cost Associated to Misclassifying the Positive and the Negative Class:* to compensate for the imbalance ratio of the two classes. For example, if the imbalance ratio is 1:10 in favour of the negative class, the penalty of misclassifying a positive example should be 10 times greater.

The first option has been applied in some works, following a random undersampling approach (Guo et al., 2008), but it has the problem of the loss of valuable information.

The second has been so far the most widely used, following different approaches, such as SMOTE (Synthetic Minority Oversampling Technique) (Chawla et al., 2002), a method proposed by Chawla et al. for creating 'artificial' samples for the minority class, in order to balance the amount of them with respect. However this technique is based in numerical computations, which consider different distance measures, in order to generate useful patterns (i.e. realistic or similar to the existing ones).

The third option implies using a method in which a cost can be associated to the classifier accuracy at every step. This was done for instance by Alfaro-Cid et al. in (Alfaro-Cid et al., 2007), where they used a Genetic Programming (GP) approach in which the fitness function was modified in order to consider a penalty when the classifier makes a false negative (an element from the minority class was classified as belonging to the majority class). However almost all the approaches deal with numerical (real, integer) data.

One interesting point about URL classification is that the study of the distance between URLs may be based in the distance between two strings, but Blanco et al. (Blanco et al., 2011) argues that the lexical distance between two URLs is not enough to classify them. In addition, the heuristic study of URLs for security purposes in the user side is not a novel practice. Also, the use of Blacklists (in this work, the *denied* URLs) and Whitelists (*allowed* URLs) are very extended practices. For instance, phishing is a problem of security that Sheng et al. and Khonki et al. (Khonji et al., 2011) tried to solve. The first work uses Blacklists as reference to avoid phishing attacks made by e-mail; the second one aims for an heuristic analysis of the URLs domain names and its ranks, in a way that a phished URL can be detected.

Also, doing some web searching we have found that a lot of companies stands for the use of one between Blacklist and Whitelist[3]. While whitelisting is

[3] http://kevtownsend.wordpress.com/2011/08/24/whitelisting-

the more restrictive solution and therefore the more secure, we think that the best solution is to use both, and for this reason the set of rules that we used covers a succession of either allowed and denied web sites.

What refers to the used techniques, DM, as well as ML, has been used since long ago in many scientific fields, and given that research in computer security was growing since the eighties (Anderson, 1980), it was in the nineties when these techniques began to be applied to security issues (Clifton and Marks, 1996).

On the one hand, DM helped to develop new solutions to computer forensics (de Vel et al., 2001), being the researchers able to extract information from large files with events gathered from infected computers. Another important advance took place after the 9/11 events, when *clustering techniques* and *social network analysis* started to be performed in order to detect pontential crime networks (Chen et al., 2003). On the other hand, and more focused on the user side like our approach, there exist some user-centric solutions to problems like user authentication in a personal device, who Greenstadt and Beal (Greenstadt and Beal, 2008) proposed to address using collected user biometrics along with machine learning techniques.

Then, when a Information Security Policy (ISP) is going to be applied, P.G. Kelley et al. (Kelley et al., 2008) found important to include the user in the machine learning process for refining the policy model. They called it *user-controllable policy learning*. Another approach to the refinement of user's privacy policies has been described by Danezis in (Danezis, 2009), for he uses ML techniques over the user's settings in a social network, being capable of restricting permissions to other people depending on their interaction with the user.

In the same line, Lim et al. propose a system (Lim et al., 2008b; Lim et al., 2008a) that evolves a set of computer security policies by means of GP, taking again into account the user's feedback. Furthermore, Suarez-Tangil et al. (Suarez-Tangil et al., 2009) take the same approach as Lim et al., but also bringing event correlation in. These two latter author's works are interesting for ours, though they are not focused on company ISPs - for instance, our case with the allowed or denied http requests -.

Finally, a system named MUSES (from Multiplatform Usable Endpoint Security System) (Mora et al., 2014) is being developed under the European Seventh Framework Programme (FP7). This system will include event treatment on the user actions inside a company, DM techniques for applying the set of policies from the company ISP to the actions, allowing or denying them, and ML techniques for improving the set of rules derived from these policies, according to user's feedback and behaviour after the system decisions (Seigneur et al., 2013).

vs-blacklisting/

3 PROBLEM AND DATA DESCRIPTION

The problem to solve is related with the application of corporate security policies in order to deal with potential URL accesses inside an enterprise. To this end a dataset of URL sessions (requests and accesses) is analysed. These data are labelled with the corresponding permission or not for that access following the aforementioned rules. The problem is then transformed into a classification one, in which every new URL request will be classified, and thus, a grant or deny action will be assigned to that pattern.

The analysed data come from an `access.log` of the Squid proxy application (Team, 2013a), in a real Spanish company. This open source tool works as a proxy, but with the advantage of storing a cache of recent transactions so future requests may be answered without asking the origin server again (Wessels, 2004). Every pattern, namely a URL session has ten variables associated, which we describe in Table 1, indicating if the variable is numeric or nominal/categorical.

The dependent variable or class is a label which inherently assigns an decision (and so the following action) to every request. This can be: *ALLOW* if the access is permitted according to the CSPs, or can be *DENY*, if the connection is not permitted. These patterns are labelled using an 'engine' based in a set of security rules, that specify the decision to make. This process is described in Subsection 4.1.

These data were gathered along a period of two hours, from 8.30 to 10.30 am (30 minutes after the work started), monitoring the activity of all the employees in a medium-size Spanish company (80-100 people), obtaining 100000 patterns. We consider this dataset as quite complete because it contains a very diverse amount of connection patterns, going from personal (traditionally addressed at the first hour of work) to professional issues (the rest of the day). Moreover, the results derived from the experiments (described in Section 5) show that this quantity of data might be big enough, but a more accurate outcome would be given with, for instance, a 24 hours long log.

Table 1: Independent Variables corresponding to a URL session (a connection to a URL for some time). The URLs are parsed as detailed in Subsection 4.2.

Variable name	Description	Type	Rank/Number of Values (if categorical)
http_reply_code	Status of the server response	Categorical	20 values
http_method	Desired action to be performed	Categorical	6 values
duration_milliseconds	Session duration	Numerical	integer in [0,357170]
content_type	Media type of the entity-body sent to the recipient	Categorical	11 values (main content), 85 values (whole content)
server_or_cache_address	IP address	Categorical	2343 values
time	connection hour (in the day)	Date	00:00:00 to 23:59:59
squid_hierarchy	It indicates how the next-hop cache was selected	Categorical	3 values
bytes	Number of transferred bytes during the session	Numerical	integer in [0,85135242]
client_address	IP address	Categorical	105 values
URL	Core domain of the URL, not taking into account the TLD	Categorical	976 values

4 METHODOLOGY

Before classification techniques are applied, a data preprocessing step has been performed. First, the raw dataset is labelled according a set of *initial corporate security rules*, i.e. every pattern is assigned to a label indication if the corresponding URL request/access would be ALLOWED or DENIED considering these rules. This step is necessary in order to transform the problem into a classification one. However, in order to apply the rules they must be transformed from their initial format into another one that can be applied in our programs (a hash in Perl[4]). This is described in Subsection 4.1.

Subsection 4.2 details how the patterns of the navigation data log (URL sessions) are also converted to a Perl hash to perform the matting/labelling process.

At the end of these two steps, the two hashes are compared in order to obtain which entries of the log should be ALLOW or DENY, know as the *labelling* step. This is similar to perform a decision process in a security system. This step results in that there are 38972 pattern belonging to class ALLOW (positive class) and 18530 of class DENY (negative class), so just a 67.78% of the samples belong to the majority class. This represents a very important problem, since a classifier that is trained considering these proportions is supposed to classify all the samples as ALLOW, getting a theoretically quite good classification accuracy equal or greater than 68%. However, in section 5 we will see that, despite the fact that some denied patterns are classified as allow, the overall performance of the classifiers are better than the expected.

Given that the dataset contains a majority of categorical/nominal data, we have performed different approaches for data balancing:

- Undersampling: we will remove random samples of the majority class until the amount in both classes are similar.
- Oversampling: we will duplicate random samples of the minority class, in order to get a close number of patterns in both classes. This has to be done due to the impossibility of creating synthetic data when dealing with categorical values (there is not a proper distance measure between two values in a category). Actually, since the number of samples in the majority class is almost twice the minority one, we have just duplicated all of those belonging to the minority class.

Finally, in Subsection 4.3 we explain the selection of the methods to apply in order to classify the data. We just have considered the patterns correctly labelled in the preprocessing phase. Thus, a supervised classification process (MacQueen et al., 1967) has been conducted on the balanced datasets. Weka Data Mining Software[5] has been used, in order to select the best set of methods in order to deal with these data. These classifiers will be further tested in Section 5.

4.1 Security Rules Parsing

In this work we have considered Drools (Team, 2013c) as the tool to create, and therefore, manage rules in a business environment. This so called Business Rule Management System (BRMS) has been developed by the JBoss community under an Apache License and it is written in Java. Though this platform consist of many components, here we focus on Drools Expert and the Drools Rule Language (DRL, (Team, 2013b)). Then, the defined rules for a certain company are inside of a file with a `.drl` extension, the file that needs to be parsed to obtain the final set of rules. In Figure 1, (a), there is the typical

[4]A *hash* in Perl is an object that represents a *hash table*, which is a set of pairs key-value. Sometimes, the value can be another hash itself.

[5]http://www.cs.waikato.ac.nz/ml/weka/

rule syntax in DRL. Two main things should be obtained from the parsing method: both left and right sides of the rule, taking into account that the left side is where the company specifies the conditions required to apply the action indicated in the right side. Also, for describing the conditions, Squid syntax is used (see Section 3), having thus the following structure: `squid:Squid(conditions)`. Finally, from the right side of the rule, the *ALLOW* or *DENY* label to apply on the data that matches with the conditions, will be extracted. The Perl parser that we have implemented applies two regular expressions, one for each side of the rule, and returns a hash with all the rules with the conditions and actions defined. The 'before and after' performing the parsing over the `.drl` file is in Figure 1.

```
rule "name"
  attributes
  when
     /* Left
     Side of the
     Rule */
   then
     /* Right
     Side of the
     Rule */
end
```

(a) Drools Rule

```
%rules = (
  rule =>{
     field => xxx
     relation =>
     xxx
     value => xxx
     action =>
     [allow,
     deny]
  },
);
```

(b) Hash Rule

Figure 1: (a) Structure of a rule in Drools Expert. (b) Resulting rule, after the parsing, in a global hash of rules.

4.2 URL Log Data Parsing

Usually, the instances of a log file have a number of fields, in order to have a registration of the client who asks for a resource, the time of the day when the request is made, and so on. In this case, we have worked with an *access.log* (see Section 3) file, converted into a CSV format file so it could be parsed and transformed in another hash of data. All ten fields of the Squid log yield a hash like the one depicted in Fig. 2.

Once the two hashes of data were created, they were compared in such a way that for each rule in the hash of rules, it was determined how many entries in the data log hash are covered by the rule, and so they were applied the label that appears as 'action' in the rule.

One of the problems was to extract from a whole URL the part that was more interesting for our purposes. It is important to point out that in a log with thousands of entries, an enormous variety of URLs can be found, since some can belong to advertisements, images, videos, or even some others does not have a domain name but are given directly by an IP address. For this reason, we have taken into account that for a domain name, many subdomains (separated by dots) could be considered, and their hierarchy grows from the right towards the left. The highest level of the domain name space is the Top-Level Domain (TLD) at the right-most part of the domain name, divided itself in country code TLDs and generic TLDs. Then, a domain and a number of subdomains follow the TLD (again, from right to left). In this way, the URLs in the used log are such as *http://subdomain...subdomain.domain.TLD/other_subdirectories*. However, for the ARFF[6] file to be created, only the domain (without the subdomains and the TLD) should be considered, because there are too many different URLs to take into consideration. Hence, applying another regular expression, the data parser implemented in Perl obtains all the core domains of the URLs, which makes 976 domains in total.

```
%logdata = (
   entry =>{
      http_reply_code => xxx
      http_method => xxx
      duration_miliseconds => xxx
      content_type => xxx
      server_or_cache_address => xxx
      time => xxx
      squid_hierarchy => xxx
      bytes => xxx
      url => xxx
      client_address => xxx
   },
);
```

Figure 2: Perl hash with an example entry. The actual hash used for this work has a total of 100000 entries, with more than a half labelled as *ALLOW* or *DENY* after the comparing process.

4.3 Classification Methods

As said in Section 3, the data used for this work is not only numerical or nominal, thus, only classification algorithms that support both types of data have been considered. Weka has a great number of possible algorithms to work with, so we have conducted a preselection phase trying to choose those which would yield better results in the experiments. More specifically, we have focused on rule-based and decision-tree-based algorithms.

In this way, a decision-tree algorithm is a group

[6]Format of Weka files

of conditions organised in a top-down recursive manner in a way that a class is assigned following a path of conditions, from the root of the tree to one of its leaves. Generally speaking, the possible classes to choose are mutually exclusive. Furthermore, these algorithms are also called "divide-and-conquer" algorithms. On the other hand, there are the "separate-and-conquer" algorithms, which work creating rules one at a time, then the instances covered by the created rule are removed and the next rule is generated from the remaining instances.

A reference to each Weka classifier can be found at (Frank and Witten, 2011). Below are described the top five techniques, obtained from the best results (See Table 2) of the experiments done in this stage, along with more specific bibliography. Nave Bayes method (Domingos and Pazzani, 1997) has been included as a baseline, normally used in text categorization problems. According to the results, the five selected classifiers are much better than this method.

Table 2: Results of all the tested classification methods on balanced data. The best ones are marked in boldface.

	Undersampling	Oversampling
Nave Bayes	91.12	91.77
Conjunctive Rule	60.14	60.02
Decision Table	94.08	90.29
DTNB	94.75	95.65
JRip	90.08	92.47
NNge	**96.49**	**98.76**
One R	93.45	93.70
PART	**96.45**	**97.54**
Ridor	87.22	89.87
Zero R	51.39	51.26
AD Tree	77.73	77.68
Decision Stump	60.14	60.02
J48	**97.02**	**98.00**
LAD Tree	79.95	79.97
Random Forest	**96.87**	**98.84**
Random Tree	95.14	98.35
REP Tree	**96.79**	**97.67**

J48. This classifier generates a pruned or unpruned C4.5 decision tree. Described for the first time in 1993 by (Quinlan, 1993), this machine learning method builds a decision tree selecting, for each node, the best attribute for splitting and create the next nodes. An attribute is selected as 'the best' by evaluating the difference in entropy (information gain) resulting from choosing that attribute for splitting the data. In this way, the tree continues to grow till there are not attributes anymore for further splitting, meaning that the resulting nodes are instances of single classes.

Random Forest. This manner of building a decision tree can be seen as a randomization of the previous C4.5 process. It was stated by (Breiman, 2001) and consist of, instead of choosing 'the best' attribute, the algorithm randomly chooses one between a group of attributes from the top ones. The size of this group is customizable in Weka.

REP Tree. Is another kind of decision tree, it means Reduced Error Pruning Tree. Originally stated by (Quinlan, 1987), this method builds a decision tree using information gain, like C4.5, and then prunes it using reduced-error pruning. That means that the training dataset is divided in two parts: one devoted to make the tree grow and another for pruning. For every subtree (not a class/leaf) in the tree, it is replaced by the best possible leaf in the pruning three and then it is tested with the test dataset if the made prune has improved the results. A deep analysis about this technique and its variants can be found in (Elomaa and Kaariainen, 2001).

NNge. Nearest-Neighbor machine learning method of generating rules using non-nested generalised exemplars, i.e., the so called 'hyperrectangles' for being multidimensional rectangular regions of attribute space (Martin, 1995). The NNge algorithm builds a ruleset from the creation of this hyperrectangles. They are non-nested (overlapping is not permitted), which means that the algorithm checks, when a proposed new hyperrectangle created from a new generalisation, if it has conflicts with any region of the attribute space. This is done in order to avoid that an example is covered by more than one rule (two or more).

PART. It comes from 'partial' decision trees, for it builds its rule set from them (Frank and Witten, 1998). The way of generating a partial decision tree is a combination of the two aforementioned strategies "divide-and-conquer" and "separate-and-conquer", gaining then flexibility and speed. When a tree begins to grow, the node with lowest information gain is the chosen one for starting to expand. When a subtree is complete (it has reached its leaves), its substitution by a single leaf is considered. At the end the algorithm obtains a partial decision tree instead of a fully explored one, because the leafs with largest coverage become rules and some subtrees are thus discarded.

These methods will be deeply tested on the dataset (balanced and unbalanced) in the following section.

5 EXPERIMENTS AND RESULTS

Several experiments have been conducted, once a subset of classification methods has been chosen in previous section. To this end, some training and test datasets have been created from the set of labelled patterns. It contains 57502 samples, with 38972 belonging to class ALLOW and 18530 to class DENY.

In order to better test the methods, two different divisions (training-test) have been done, namely 90%-10% and 80%-20%. Moreover, two additional splits have been considered in every case, using both a random and a sequential approach for selecting samples from the original file. Thus, in the latter, consecutive patterns have been included in the training file up to the desired percentage. The rest have composed the test file. In the first approach, a random selection is performed.

The aim of the sequential division is to compare if the online activity of the employees considering URL sessions could be somehow 'predicted', just using data from previous minutes or hours.

With respect to the data, the initial file was unbalanced, as it can be seen in the number of patterns per class. Hence, as stated in Section 3, two data balancing methods have been applied to all the files, to get similar numbers in both classes: undersampling (random removal of ALLOW patterns) and oversampling (duplication of DENY patterns).

Results for unbalanced data are presented in Table 3. Three different tests have been done for the random pattern distribution approach, so the mean and standard deviation are shown in the corresponding columns.

As it can be seen, all the five methods achieved a high performance classifying in the right way the test dataset. Also, these results are not like this by chance, as shown by a low standard deviation. Although it was expected that the results from the 90%-10% division were slightly better, in the future a more aggressive division will be executed so the methods can be really proved with much less training data.

What matters to the results of the experiments made with the sequential data, they are worse than the obtained from the random data, but still they are good ($> 85\%$). This is due to the occurrence of new patterns from a certain time (maybe there are some requests that are made just at one specific time in a day, or in settled days), and then there is no sufficient similarity between the training data and the classifying of the test data set may fail. The loss of 5 to 6 points in the results of the 90%-10% division is the first unexpected or unlogical result of the experiments, but they also reinforce the previous theory.

The technique that lightly stands out over the others is *Random Forest*, being the best in almost every case, even in the experiments with the most complex sequential divisions. However, if we focus on the standard deviation, *REP Tree* is the chosen one, as its results present robustness.

For its part, results obtained from unbalanced data are shown in Table 4. Again the corresponding to the random partitions come from the mean of three blocks of experiments, and so are specified the standard deviations. The Table illustrates two segments of results, obtained from the undersampled data and from the oversampled data. For each one, the 90%-10% and 80%-20% divisions were also made.

Applying Undersampling. In comparison with those results from Table 3, these go down one point (in the case of randomly made divisions) to six points (sequential divisions). The reason why this happens is that when randomly removing ALLOW patterns, we are really losing information, i. e. key patterns that could be decisive in a good classification of a certain set of test patterns.

Applying Oversampling. Here we have duplicated the DENY patterns so their number could be up to that of the ALLOW patterns. However, it does not work as well as in other approaches which uses numerical computations for creating the new patterns to include in the minority class. Consequently, the results have been decreased.

In both cases it is noticeable that taking the data in a sequential way, instead of randomly, lower the results. It is clear that due to the fact that performing undersampling some patterns are lost while in the case of oversampling they all remain, *undersampling results* are better. Then, in this case the algorithm with best performance is *J48*, though *Random Forest* follows its results very closely in random datasets processing, and *REP Tree*, which is better than the rest when working with sequential data. Nevertheless, generally speaking and given the aforementioned reasons, performing data balancing methods yields worse results.

Furthermore, we have found that for the data sets taken consecutively, the methods always classify worse the DENY labels, as they label them as ALLOW patterns. This is worth further study because it is the worst situation. It would be preferable to have a false positive in a DENY pattern, rather than a false negative and permit a request that is forbidden in the ISP.

Regarding the obtained rules/trees, we want to remark that the majority are based on the URL in order to discriminate between the two classes, how-

Table 3: Percentage of correctly classified patterns for non-balanced data.

	80% Training - 20% Test		90% Training - 10% Test	
	Random (mean)	Sequential	Random (mean)	Sequential
J48	97.56 ± 0.20	88.48	97.70 ± 0.15	82.28
Random Forest	97.68 ± 0.20	89.77	97.63 ± 0.13	82.59
REP Tree	97.47 ± 0.11	88.34	97.57 ± 0.01	83.20
NNge	97.23 ± 0.10	84.41	97.38 ± 0.36	80.34
PART	97.06 ± 0.19	89.11	97.40 ± 0.16	84.17

Table 4: Percentage of correctly classified patterns for balanced data (under- and oversampling).

	80% Training - 20% Test				90% Training - 10% Test			
	Undersampling		Oversampling		Undersampling		Oversampling	
	Rand (mean)	Sequential	Rand (mean)	Sequential	Rand (mean)	Sequential	Rand (mean)	Sequential
J48	97.05 ± 0.25	84.29	97.40 ± 0.03	85.66	96.85 ± 0.35	76.44	97.37 ± 0.06	74.24
Random Forest	96.61 ± 0.17	88.59	97.16 ± 0.19	89.03	96.99 ± 0.13	79.98	97.25 ± 0.33	81.33
REP Tree	96.52 ± 0.13	85.54	97.13 ± 0.25	85.41	96.55 ± 0.10	77.65	97.14 ± 0.09	76.81
NNge	96.56 ± 0.42	85.28	96.90 ± 0.28	83.46	96.33 ± 0.05	81.93	96.91 ± 0.06	78.73
PART	96.19 ± 0.14	85.16	96.82 ± 0.09	84.50	96.09 ± 0.10	79.70	96.68 ± 0.11	78.16

ever we also found several ones which consider variables/features different of this to make the decision. For instance:

```
IF server_or_cache_address = "90.84.53.17"
THEN DENY

IF server_or_cache_address = "173.194.78.103"
THEN ALLOW

IF content_type =
 "application/vnd.google.safebrowsing-update"
THEN DENY

IF server_or_cache_address = "173.194.78.94"
AND content_type_MCT = "text"
AND content_type = "text/html"
AND http_reply_code = "200"
AND bytes > 772
THEN ALLOW

IF server_or_cache_address = "173.194.34.225"
AND http_method = "GET"
AND duration_milliseconds > 52
THEN ALLOW

IF server_or_cache_address = "90.84.53.49"
AND time <= 33758000
THEN ALLOW
```

These are the interesting rules for our purposes, since they are somehow independent of the URL to which the client requests to access. Thus, it would be potentially possible to allow or deny the access to unknown URLs just taking into account some parameters of the session.

Of course, some of these features depend on the session itself, i.e. they will be computed after the session is over, but the idea in that case would be 'to refine' somehow the existing set of URLs in the White List. Thus, when a client requests access to a Whitelisted URL, this will be allow, but after the session is over, and depending on the obtained values, and on one of these classifiers, the URL could be labelled as DENIED for further requests. This could be a useful decision-aid tool for the CSO in a company, for instance. In the case that the features considered in the rule can be known in advance, such as `http_method`, or `server_or_cache_address`, for instance, the decision could be made in real-time, and thus, a granted URL (Whitelisted) could be DENIED or the other way round.

The tree-based methods also yield several useful branches in this sense, but they have not been plotted here because of the difficulty for showing/visualizing them properly.

6 CONCLUSIONS AND FUTURE WORK

In this paper a set of classification methods have been applied in order to perform a decision process inside a company, according to some predefined corporate security policies. This decision is focused on allowing or denying URL access requests, but just considering previous decisions on similar requests, not having specific rules in a White/Black List, defined for those URLs. Thus, the proposed method could allow or deny an access to a URL based in additional terms rather than just the specific URL string. This could be very useful since new URLs could be automatically 'Whitelisted' or 'Blacklisted', just depending on some of the connection parameters, such as the `content_type` of the access or the `IP` of the client which makes the request.

To this aim, we have started from a big dataset (100000 patterns) about employees' URL sessions information, and considering a set of URL access permissions, we have composed a labelled dataset

(57000 patterns). Over that set of data, we have tested several classification methods, after some data balancing techniques have been applied. Then, the best five have been deeply proved over several training and test divisions, and with two methods: using sequential patterns (consecutive URL accesses), and taking them in a randomly way.

The results show that classification accuracies are between 95% and 97%, even when using the unbalanced datasets. However, they have been diminished because of the possible loss of data that comes from performing an undersampling (removing patterns) method; or taking the training and the data sets in a sequential way from the main log file, due to the fact that certain URL requests can be made only at a certain time.

In this way, we can conclude that the approach has been successful and it would be a useful tool in an enterprise.

Future lines of work include conducting a deeper set of experiments trying to test the generalisation power of the method, maybe considering bigger data divisions, bigger data sets (from a whole day or working day), or adding some kind of 'noise' to the dataset. So that, considering the good classification results obtained in this work, the next step could be the application of these methods in the real system from which data was gathered, counting with the opinion of expert CSOs, in order to know the real value of the proposal. The study of other classification methods could be another research branch, along with the implementation of a Genetic Programming approach, which could deal with the imbalance problem using a modification of the cost associated to misclassifying, could be done (as the authors did in (Alfaro-Cid et al., 2007)).

Finally, we also point to extract additional information from the URL string, than could be transformed into additional features that could be more discriminative than the current set. Moreover, a data process involving summarizing data about sessions (such as number of requests per client, or average time connection) will be also considered.

ACKNOWLEDGEMENTS

This paper has been funded in part by European project MUSES (FP7-318508), along with Spanish National project TIN2011-28627-C04-02 (ANYSELF), project P08-TIC-03903 (EVORQ) awarded by the Andalusian Regional Government, and projects 83 (CANUBE), and GENIL PYR-2014-17, both awarded by the CEI-BioTIC UGR.

REFERENCES

Alfaro-Cid, E., Sharman, K., and Esparcia-Alczar, A. (2007). A genetic programming approach for bankruptcy prediction using a highly unbalanced database. In Giacobini, M., editor, *Applications of Evolutionary Computing*, volume 4448 of *Lecture Notes in Computer Science*, pages 169–178. Springer Berlin Heidelberg.

Anderson, A. J. P. (1980). Computer security threat monitoring and surveillance. Technical report, James P. Anderson Co., Fort Washington, PA.

Blanco, L., Dalvi, N., and Machanavajjhala, A. (2011). Highly efficient algorithms for structural clustering of large websites. In *WWW '11 Proceedings of the 20th international conference on World wide web.*, pages 437–446. ACM.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Chawla, N. (2005). Data mining for imbalanced datasets: An overview. In Maimon, O. and Rokach, L., editors, *Data Mining and Knowledge Discovery Handbook*, pages 853–867. Springer US.

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: Synthetic minority oversampling technique. *J. Artif. Int. Res.*, 16(1):321–357.

Chen, H., Chung, W., Qin, Y., Chau, M., Xu, J. J., Wang, G., Zheng, R., and Atabakhsh, H. (2003). Crime data mining: An overview and case studies. In *Proceedings of the 3rd National Conference for Digital Government Research (dg.o 2003)*, volume 130, pages 1–5. Digital Government Society of North America.

Clifton, C. and Marks, D. (1996). Security and privacy implications of data mining. In *ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, pages 15–19.

Danezis, G. (2009). Inferring privacy policies for social networking services. In *Proceedings of the 2Nd ACM Workshop on Security and Artificial Intelligence*, AISec '09, pages 5–10, New York, NY, USA. ACM.

de Vel, O., Anderson, A., Corney, M., and Mohay, G. (2001). Mining e-mail content for author identification forensics. *SIGMOD Record*, 30(4):55–64.

Domingos, P. and Pazzani, M. (1997). On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29:103–137.

Elomaa, T. and Kaariainen, M. (2001). An analysis of reduced error pruning. *Artificial Intelligence Research*, 15(-):163–187.

Frank, E. and Witten, I. H. (1998). Generating accurate rule sets without global optimization. In Shavlik, J., editor, *Fifteenth International Conference on Machine Learning*, pages 144–151. Morgan Kaufmann.

Frank, E. and Witten, I. H. (2011). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers, third edition.

Greenstadt, R. and Beal, J. (2008). Cognitive security for personal devices. In *Proceedings of the 1st ACM Workshop on Workshop on AISec*, AISec '08, pages 27–30, New York, NY, USA. ACM.

Guo, X., Yin, Y., Dong, C., Yang, G., and Zhou, G. (2008). On the class imbalance problem. In *Natural Computation, 2008. ICNC '08. Fourth International Conference on*, volume 4, pages 192–201.

Japkowicz, N. and Stephen, S. (2002). The class imbalance problem: A systematic study. *Intell. Data Anal.*, 6(5):429–449.

Kelley, P. G., Hankes Drielsma, P., Sadeh, N., and Cranor, L. F. (2008). User-controllable learning of security and privacy policies. In *Proceedings of the 1st ACM Workshop on Workshop on AISec*, AISec '08, pages 11–18, New York, NY, USA. ACM.

Khonji, M., Jones, A., and Iraqi, Y. (2011). A novel phishing classification based on url features. In *GCC Conference and Exhibition (GCC)*, pages 221–224. IEE.

Lim, Y. T., Cheng, P. C., Clark, J., and Rohatgi, P. (2008a). Policy evolution with genetic programming: A comparison of three approaches. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 1792–1800.

Lim, Y. T., Cheng, P. C., Rohatgi, P., and Clark, J. A. (2008b). Mls security policy evolution with genetic programming. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, GECCO '08, pages 1571–1578, New York, NY, USA. ACM.

MacQueen, J. et al. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, number 14, pages 281–297. California, USA.

Martin, B. (1995). Instance-based learning: Nearest neighbor with generalization. Master's thesis, University of Waikato, Hamilton, New Zealand.

Mora, A., De las Cuevas, P., Merelo, J., Zamarripa, S., Juan, M., Esparcia-Alczar, A., Burvall, M., Arfwedson, H., and Hodaie, Z. (2014). MUSES: A corporate user-centric system which applies computational intelligence methods. In et al., D. S., editor, *29th Symposium On Applied Computing*, pages 1719–1723.

Oppliger, R. (2011). Security and privacy in an online world. *IEEE Computer*, 44(9):21–22.

Quinlan, J. R. (1987). Simplifying decision trees. *Man-Machine Studies*, 27(3):221–234.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA.

Seigneur, J.-M., Kölndorfer, P., Busch, M., and Hochleitner, C. (2013). A Survey of Trust and Risk Metrics for a BYOD Mobile Working World. In *Third International Conference on Social Eco-Informatics*.

Suarez-Tangil, G., Palomar, E., Fuentes, J., Blasco, J., and Ribagorda, A. (2009). Automatic rule generation based on genetic programming for event correlation. In Herrero, l., Gastaldo, P., Zunino, R., and Corchado, E., editors, *Computational Intelligence in Security for Information Systems*, volume 63 of *Advances in Intelligent and Soft Computing*, pages 127–134. Springer Berlin Heidelberg.

Team, S. (2013a). Squid website.

Team, T. J. D. (2013b). Drools documentation. version 6.0.1.final.

Team, T. J. D. (2013c). Drools website.

Wessels, D. (2004). *Squid: The Definitive Guide*. O'Reilly Media, Inc., 1 edition.