A COMPARATIVE STUDY OF DATA MINING ALGORITHMS FOR NETWORK INTRUSION DETECTION IN THE PRESENCE OF POOR QUALITY DATA

(complete-paper)

Eitel J.M. Lauría
Marist College
Eitel.Lauria@Marist.edu

Giri K. Tayi
University at Albany, SUNY
g.tayi@albany.edu

Abstract: In this paper we discuss our research in applying classification methods for computer network intrusion detection. Using two different algorithms for classification (decision trees and naive Bayes classifier) we build a predictive model capable of distinguishing between "bad" TCP/IP connections, called intrusions or attacks, and "good" normal TCP/IP connections. We investigate the effect of training the models using both clean and dirty data. Our purpose is to analyze the predictive power of the network intrusion classification models under circumstances in which training data quality is at issue.

Key Words: Data mining, Data Quality, Network Intrusion Detection, Classification

INTRODUCTION

If data is characterized as recorded facts, then information is the set of patterns or expectations that underlie the data [17]. But information is not packaged in a standard easy-to-retrieve format. It is an underlying and usually subtle and misleading concept buried in massive amounts of raw data [7]. Modern organizations are overloaded with data, and don't have the time to look at it. They must find ways to automatically analyze it, summarize it, classify it, and provide for the discovery of hidden trends, patterns and anomalies. This is the goal of a set of methods, technologies and tools drawn from statistics, computer science, artificial intelligence and machine learning, that have come to be known as data mining and knowledge discovery.

Data mining has been successfully applied to different activities and scenarios, including scientific research (genomic data mining, signal recognition in astronomy, medical diagnosis), and business areas such as manufacturing, finance and marketing. Computerization of business processes leads to abundance of raw data, which in turn facilitates the applicability of data mining techniques. Data mining is being used in marketing to target customer needs, increase response rate of direct-mail campaigns, and improve customer retention. Financial institutions use data mining to analyze and predict market fluctuations based on historical times series and real-time data. Credit card issuers and insurance companies have pioneered the use of data mining with the purpose of detecting fraud in purchases and claims.

Recently there has been much interest in applying data mining to computer network intrusion detection. An intrusion can be defined as "any set of actions that attempt to compromise the integrity, confidentiality or availability of a resource" [5]. A network intrusion attack can compromise the stability or the security of the information stored in those computers connected to it. Considering that in today's world, companies depend on their physical networks and on their level of connectivity to survive, it comes at no surprise that the role of network intrusion detection has grown so rapidly. While there may still be different approaches as to what is the best way to protect a company's network, it is evident that an intrusion detection system is certainly an important asset among the group of tools used to secure the network architecture. Added security measures have failed in many cases to stop the wide variety of possible attacks. The goal of intrusion detection is to build a system that would automatically scan network activity and detect such intrusion attacks, providing the necessary information to the system administrator to allow for corrective action [11]. A strong case can be made for the use of data mining techniques to improve the current state of intrusion detection. Stolfo and Lee [9] describe several approaches to data mining, which are particularly useful in this domain:

- **Association:** determines relations between fields in the database. Finding out the correlations in audit data will provide insight for selecting the right set of system features for intrusion detection;
- **Sequence analysis:** models sequential patterns. These algorithms can help us understand what (time-based) sequence of audit events are frequently encountered together;
- **Classification:** mapping a data item into one of several pre-defined categories. An ideal application in intrusion detection will be to train a program by using sufficient "normal" and "abnormal" audit data, then apply the program to determine (future) audit data as belonging to the normal class or the abnormal class. Classification is sometimes called supervised learning, because the learning algorithm operates under supervision by being provided with the actual outcome for each of the training examples (also known as labeled examples).

Although the data mining techniques may perform quite efficiently on test data, real world applications of data mining necessarily involves use of data whose quality varies and hence could be poor on some dimensions and excellent on others. Thus the efficacy and usability of a data mining technique could be strongly influenced by the quality of data available to an organization. It has been well recognized that organizational databases and data sources have persistent data quality problems [15]. Data consumers view data quality to be a composite of several dimensions such as accuracy, believability, completeness, interpretability [14]. So a database that is excellent in terms of completeness may in fact be of poor quality because of the presence of erroneous data. So for any data mining effort to be successful it should be preceded by a data quality enhancement activity. However, determining which dimension of data quality should be improved and to what extent in accomplishing a successful data mining application is not straightforward. It requires understanding the interaction between the problem domain, the nature of data errors in that domain and the characteristics of the proposed data mining technique.

In this paper, we carryout a comparative study of two classification techniques in the domain of network intrusion detection when the audit data could be "dirty" or of poor quality. As discussed by [8], a high statistical accuracy should not be the main goal of an intrusion detection system; rather, the more important goal should be the maximum reduction in intrusion damage cost with minimum intrusion detection operational cost. Investing in network intrusion detection should help maximize the user network security goals, while minimizing the operational costs. Fan et al [2] examine some of the relevant factors, models and metrics related to intrusion detection systems.

A very important aspect of research in this area should then focus on the study of automated techniques for building intrusion detection systems that are optimized for user-defined cost metrics. Given the fact that the time and resources available to deliver clean audit data are limited, the problem can be seen from the perspective of implementing data mining techniques robust enough to cope with dirty data, and

provide adequate automated intrusion detection mechanisms.

In a typical networking environment, training data used for classification purposes is usually quite accurate, as it is being automatically collected from the audit files recorded by the operating system software. The weak link is found to be in this case in the process of labeling the training examples. This may be given to different factors including errors in label data entry and lack of precision in expert judgment. This paper investigates the application of classification to the task of computer network intrusion detection, and the extent to which errors in the labeling of the training data records affect the classification models. The datasets on which these techniques were applied were extracted from data files used for The Third International Knowledge Discovery and Data Mining Tools Competition, which was held in conjunction with KDD-99 (The Fifth International Conference on Knowledge Discovery and Data Mining). The competition task was to build a network intrusion detector, a predictive model capable of discriminating normal TCP/IP connections from intrusions or attacks, and further classifying those attacks into a set of previously identified classes. In this paper we replicated the experiment on a reduced sample data set, using two different algorithms for classification: a C4.5 decision tree and a naive Bayes learner. As the tool of choice, we used Weka [16], a data mining toolkit developed at the University of Waikato, New Zealand.

We start by providing an introduction to the classifying algorithms under consideration, namely decision trees and naive Bayes classifiers. Following this, we present the different stages of the experiment and the results obtained. We finish with a discussion of the results and our conclusions.

DECISION TREES

A decision tree is a flow-chart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of a test, and leaf nodes represent classes [4]. The topmost node in a tree is the root node. Consider the example in Table 1 adapted from [10]. We are dealing with records reporting on weather conditions for playing tennis. The class attribute specifies whether or not to play.

outlook	temperature	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	No

(a) List of Instances

Attribute Name	Values
outlook	sunny, overcast, rainy
temperature	hot, mild cool
humidity	high, normal
windy	true, false
play (class attribute)	yes, no

(b) List of Attributes

```
outlook = sunny
|    humidity  =  high: no
|    humidity = normal: yes
outlook = overcast: yes
outlook =  rainy
|     windy: yes
|     windy: no
```

(c) Decision Tree

Table 1: Weather Data Set

For our example we could obtain a decision tree like the one depicted in (c). This representation is equivalent to defining a set of rules. For example, according to the tree, if the outlook is sunny, and the humidity is high, we don't play. With this tree structure we could then predict future decisions, given a new instance. As we saw in (c), a path is traced from the root to a leaf node that holds the class prediction for the sample. The basic algorithm for decision tree induction is a greedy algorithm that constructs decision trees in a top-down recursive divide-and-conquer manner [4]. The algorithm uses the same process recursively to form a decision tree for the samples at each partition. We still need to deal with the issue of finding the order of precedence of the attributes. For that purpose we define an *impurity* function that measures the proportion of instances that belong to the different classes. We choose the attribute that maximizes the reduction in impurity from the root node to the children.

Since tree size is not limited in the growing process, a tree may be more complex than necessary to describe the data. Many of the branches may reflect anomalies due to outliers, missing data, etc. This may lead to overfitting the data: the tree learning process adjusts the tree structure to maximize the fit of the training data set, but by doing so it can therefore fail to classify novel patterns. It is unlikely that a complex decision boundary would provide good generalization- it seems to be tuned to the particular features of the training data set, rather than some underlying characteristics of the set of all possible instances that need to be classified[1]. Tree pruning methods address this problem of overfitting the data by considering each of the decision nodes in a tree to be candidates for pruning.

ID3, C4.5 and C5.0 are multiway classification tree algorithms introduced by Quinlan [12, 13]. These algorithms use an entropy-based impurity function as a heuristic rule for selecting the attribute that will best separate the samples into individual classes. ID3 is a rudimentary version limited to handle discrete

[1] The philosophical debate regarding this criterion has been going on for centuries. William of Occam (14th century) was one of the first thinkers to discuss the question of whether simpler hypotheses are better than complicated ones. For this reason this approach goes by the name of *Occam's razor*

predictor variables. C4.5 and its newest release, C5.0 introduce a number of extensions, among them: providing support for handling continuous predictor attributes, and sophisticated tree pruning.

NAIVE BAYES CLASSIFICATION

Bayesian methods make explicit use of probability for quantifying. Bayes theorem provides the means of calculating the probability of a hypothesis (posterior probability) based on its prior probability, the probability of the observations and the likelihood that the observational data fits the hypothesis.

$$\mathrm{P}(H \mid D) = \mathrm{P}(D \mid H) \cdot \mathrm{P}(H) / \mathrm{P}(D) \tag{1}$$

Note that $\mathrm{P}(H \mid D)$ is the probability of a certain hypothesis based on a set of observational data *D*, given a certain context (posterior probability of hypothesis *H*); $\mathrm{P}(D \mid H)$ is the likelihood of the observations given a certain hypothesis; $\mathrm{P}(H)$ is the intrinsic probability of hypothesis *H*, before considering the evidence *D* (prior probability); and $\mathrm{P}(D)$ is the probability of the observations, independent of the hypothesis, that can be interpreted as a normalizing constant. Bayes rule can then be reformulated as $\mathrm{P}(H \mid D) \propto \mathrm{P}(D \mid H) \cdot \mathrm{P}(H)$, which means that the probability of the hypothesis *H* is being updated by the likelihood of the observed data *D*.

The practical application of this expression is rather straight-forward. Given a set of hypotheses *H* and a set *D* of observational data we can estimate the most probable hypothesis *H* given *D*, by comparing different instances of the above expression for each hypothesis H and choosing the one that holds the largest posterior probability (also called maximum a posteriori probability or MAP).

$$\text{Most probable } H \equiv H_{\mathrm{MAP}} = \arg\max \left[\mathrm{P}(D \mid H) \cdot \mathrm{P}(H) \right] \tag{2}$$

Suppose we have a classification problem where the class variable is denoted by *C* and can take values $c_1, c_2, ,.. c_k$. Consider a data sample *D* represented by *m* attributes $A_1, A_2, .., A_m$ of which the observations $(a_1, a_2, ,.. a_m)$ have been taken for each instance of *D*. Suppose that each instance of the data sample *D* is classified as $c_1, c_2, ,.. c_k$. The Bayesian approach to classifying a new instance would then be to assign the most probable target value (a class value of type c_i) by calculating the posterior probability for each class given the training data set, and from them choosing the one that holds the maximum value.

$$c_{\mathrm{MAP}} = \arg\max_{c_i \in C} \left[\mathrm{P}(D \mid c_i) \cdot \mathrm{P}(c_i) \right] \tag{3}$$

When the dependency relationship among the attributes used by the classifier are unknown, a simplified approach, known as the naive Bayesian classifier criterion, is to assume that the attributes are conditionally independent given the class. In other words:

$$\mathrm{P}(D \mid c_i) = \prod_{j=1}^{m} \mathrm{P}(A_j = a_j \mid c_i) \quad , c_i \in C \tag{4}$$

The conditional probabilities of each individual attribute can be estimated from the frequency distributions of the sample data set *D* as N_{ij} / N_i, where N_{ij} is the number of training examples for which attribute $A_j = a_j$ and class value is c_i; and N_i is the number of training examples for which the class value is c_i. If the prior probabilities $\mathrm{P}(c_i)$ are not known, they can also be estimated drawing its probabilities from the sample data set frequency distributions. If the attribute values are continuous, they need to be discretized first, making some assumptions regarding the probability density functions for each of them (for more information regarding discretization procedures, see [6]).

For example, if we take the data sample reporting on weather conditions for playing tennis (presented in previous sections), we could estimate the class probabilities P(play=yes) and P(play=no) as follows:

P(play=yes) = (# of instances were play=yes) / (total # of instances) = 9/14

P(play=no) = (# of instances were play=no) / (total # of instances) = 5/14

In the case of the conditional probabilities we would compute for example:

$$\text{P(outlook=overcast |play = yes)} = \frac{\text{(\# of instances were outlook=overcast and play=yes)}}{\text{(\# of instances were play=yes)}} = \frac{2}{9}$$

Although the naive Bayes assumption may seem over simplifying, the fact is that they work quite well in certain classification problems. Various empirical studies of this classifier have rendered comparable results to those obtained by using state of the art classifiers (see [10] for more details).

ESTIMATING CLASSIFICATION ERRORS

Given a single labeled instance for which there are *k* possible class values, the classifier calculates a probability pi for each class c_i, yielding a probability vector $(p_1, p_2, .., p_k)$, the sum of which adds up to 1. The actual outcome for that instance is one of the classes; and this fact can be expressed by considering a vector $(v_1, v_2, .., v_k)$, where $v_i = 1$, the actual class, and the rest are equal to 0. As described by Witten and Frank [17], we can express the penalty of considering a probability for each class as a loss function that depends on both the *p* vector and the *v* vector. A common criteria is to calculate a quadratic loss function as:

$$\mathrm{L} = \sum_{j=1}^{k} (p_j - a_j)^2 \tag{5}$$

This expression computes the loss function for a single instance. Weka averages this expression over all the instances to provide an error measurement of the classifier, reporting the square root of the overall average loss function as the mean squared error.

EXPERIMENTAL SETUP

As stated before, we replicated, on a reduced data set, the main task of The Third International Knowledge Discovery and Data Mining Tools Competition (in KDD-99): that is, to build a network intrusion detector, a predictive model capable of distinguishing between "bad" TCP/IP connections, called intrusions or attacks, and "good" normal connections from a data file containing s a wide variety of intrusions simulated in a military network environment. We chose in this paper to focus on the single dimension of accuracy, considering different levels of quality in the training data set. We use simulation to show the effects of erroneous labeling of training data records in a context where the outcome of the analysis is a predictive model and interest is focused on the accuracy of the model for detecting network intrusions and classifying them.

Data Source

To perform the experiment, we downloaded the *corrected.gz* file from the KDD-99 web page[2], a 45 MB

[2] More details can be found at http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

data file contaning class labels used as the test file for the original competition. From the *corrected.gz* file we extracted 11000 records, 10000 for the training dataset, and 1000 for the test dataset. To avoid biasing the extracted dataset (we checked that classes were not evenly distributed in the original file), we performed random sampling on the *corrected.gz* file. The initial data file format is displayed in Table 2 together with the list of values of the class attribute (connection_type).

Attribute name	Description	Type
duration	length (number of seconds) of the connection	continuous
protocol_type	type of the protocol, e.g. tcp, udp, etc.	discrete
service	network service on the destination, e.g., http, telnet, etc.	discrete
flag	normal or error status of the connection	discrete
src_bytes	number of data bytes from source to destination	continuous
dst_bytes	number of data bytes from destination to source	continuous
land	1 if connection is from/to the same host/port; 0 otherwise	discrete
wrong_fragment	number of ``wrong'' fragments	continuous
urgent	number of urgent packets	continuous
hot	number of ``hot'' indicators	continuous
num_failed_logins	number of failed login attempts	continuous
logged_in	1 if successfully logged in; 0 otherwise	discrete
num_compromised	number of ``compromised'' conditions	continuous
root_shell	1 if root shell is obtained; 0 otherwise	discrete
su_attempted	1 if ``su root'' command attempted; 0 otherwise	discrete
num_root	number of ``root'' accesses	continuous
num_file_creations	number of file creation operations	continuous
num_shells	number of shell prompts	continuous
num_access_files	number of operations on access control files	continuous
num_outbound_cmds	number of outbound commands in an ftp session	continuous
is_hot_login	1 if the login belongs to the ``hot'' list; 0 otherwise	discrete
is_guest_login	1 if the login is a ``guest''login; 0 otherwise	discrete
count	number of connections to the same host as the current connection in the past 2 seconds	continuous
serror_rate	% of connections that have ``SYN'' errors	continuous
rerror_rate	% of connections that have ``REJ'' errors	continuous
same_srv_rate	% of connections to the same service	continuous
diff_srv_rate	% of connections to different services	continuous
srv_count	number of connections to the same service as the current connection in the past 2 seconds	continuous
srv_serror_rate	% of connections that have ``SYN'' errors	continuous
srv_rerror_rate	% of connections that have ``REJ'' errors	continuous
srv_diff_host_rate	% of connections to different hosts	continuous
connection_type	Type of connection: normal or intrusion	discrete
connection_type: apache2., back., buffer_overflow., guess_passwd., httptunnel., ipsweep., mailbomb., mscan., multihop., neptune., nmap., normal., pod., portsweep., processtable., rootkit., saint., satan., smurf., snmpgetattack., snmpguess., sqlattack., warezmaster., xlock.		

Table 2: Data File Format

As a first task previous to the classification, we ran a group of summary statistics on the dataset, in order to get a feeling on the underlying characteristics of the component attributes of the file. We performed this task on all 11000 records, disregarding the fact that some of then would then be used for training, and the rest for testing purposes. We made the following findings:

- The class attribute (connection_type) is strongly concentrated in 3 classes: neptune (2072, 18.8%), normal (2264, 20.6%) and smurf (5674, 51.6%), with the other classes accounting for the remaining 10%. This suggests that the classifier should tend to choose among these 3 categories to make its predictions.
- The attributes *num_outbound_cmd* and *land* contain only 0s and are therefore irrelevant

We also did a graphical exploration on the group of continuous attributes to check their density distribution. We could verify that of the 32 attributes checked, 19 of them approximated a gaussian distribution, although exhibiting some outliers in their density plots.

Finally, the 11000 records data set was divided into two data sets: the training data set (10000 records), and the test data set, containing the remaining 1000 records.

Dirty Data Simulation

Quality of the training data is one of the most important factors in achieving good classifier performance. Training data quality is a function of the number of examples, how representative the examples are, and the attributes used to describe them [1]. In our case none of these factors seem to be a critical issue:

- A large number of records from different log files may be consolidated into one data file rendering a training data set with enough examples to be considered statistically representative
- Data is automatically logged by the operating system. There are no typos or errors of judgment that could bias the training data set in any manner.

But classification uses labeled training examples to build a model. The labels are provided by human experts who manually review cases. This is a typical situation in which errors of judgment given by the amount of time dedicated to review each case, the resources at hand, the training and expertise of the analysts and the subtlety of the analysis may affect the quality of the training data. The training data set could therefore contain clean attributes but dirty labels.

This has been our rationale in simulating dirty training data sets. Given the initial training data set with a length of 10000 records we simulated data errors by perturbing the data in the following manner:

a. To simulate an error of judgment of the expert labeling the records we replaced an intrusion with a normal connection (normal. label) and a normal connection with one of two possible intrusions (smurf. or neptune.) with a probability given by their distribution in the original training sample. We followed this approach given the fact that these two intrusions represent roughly 89% (65% and 24% respectively) of the attacks contained in the training data set, with the remaining 11% distributed among 21 classes (the following type of attack in order of relevance holds 3% of the samples). It is reasonable to infer that a false positive error would fall into one of these two dominant types of attack
b. Using this approach we generated 13 dirty data sets with perturbations of 1%, 5%, 10%, 15%, 20%, 25%, 30%, 35%, 40%, 45%, 50%, 60% and 70% of the cases.

Model generation and performance evaluation

During the learning stage of the experiment a model was built for every combination of classifying algorithm and training data set, 2 x 14 = 28 models all in all. We applied naïve Bayes and a C4.5 decision tree classification with tree pruning on the clean training data set and on the simulated dirty data sets, which had been previously reformatted to accommodate the file requirements of Weka (Weka reads data from a text file that contains a header with the list of attributes and its data types, followed by the data structured as a comma separated file). In the case of naive Bayes, we used Weka to calculate the prior probabilities for each class together with the information used to compute the conditional probabilities, including counts of each attribute value for each class of connection (in the case of discrete attributes), and mean and standard deviation of the probability distribution of each attribute value given the connection type (in the case of continuous attributes). Weka's implementation of the C4.5 algorithm built and displayed a pruned decision tree for each training data set under consideration. The details of the experiment together with the probability tables and the decision trees generated in each run are available from the authors.

To predict the performance of the classifiers on new data we applied two different procedures:

a) We used the test data set that had been randomly extracted from the original sample of 11000 records. Records were extracted by applying stratified sampling, to avoid uneven representation of the class values in the test data set relative to the clean training data set. We checked the frequency distribution of the class values in the test set and verified that they matched the distribution of the

training data set (see Table 3). Each of the 28 models built were tested, and the predictive performance of each model was measured: we computed the accuracy as a mean value and a 95% confidence interval; the mean root squared error (the square root of the quadratic loss function); and the Kappa coefficient, which represents the proportion of agreement obtained after removing the proportion of agreement that could be expected to occur by chance [3].

Class values	Clean training data set	Test data set
smurf.	51.47	52.70
normal.	20.54	21.00
neptune.	18.98	17.40
Others	9.01	8.90

Table 3. Class value distribution

b) We performed stratified 10-fold cross-validation (*n*-fold cross-validation with $n = 10$) on each of training data sets for each of the two classifiers under analysis. As noted by [17], 10-fold cross-validation has become in practice the standard way of assessing the performance of a classifier. In this statistical procedure the data is split into a fixed number (*n*) of partitions (folds) of approximately equal size, in each of which the class is represented in approximately the same proportions as in the full dataset, and each in turn is used for testing while the remainder is used for training. The procedure is repeated *n* times, to guarantee that every instance participates one time in the testing process so that in the end every instance has been used one time for testing purposes. The accuracy metrics are computed for each holdout and the estimates are averaged to yield and overall estimate of these metrics. As before, the following metrics were calculated: accuracy as a mean value and an error bar with a confidence of 95%; the mean root squared error; and the Kappa coefficient.

Results

The results of the classification analysis are presented in Table 4, Figure 1 and Figure 2. Table 4 shows the assessment of accuracy performance of both classifiers, using 10-fold cross-validation and the test data set, for different percentages of label errors. The accuracy is provided as a point estimate +/- a 95% confidence boundary. Figure 1 displays the mean accuracy of the classifiers as a function of the percentage of label errors. Figure 2 graphs the mean squared error of the classifiers for each test.
It can be seen that for clean training data the C4.5 algorithm outperforms the naïve Bayes classifier, attaining an accuracy in the interval (0.9653,0.9847) with the test data set, as compared to (0.9021,0.9359) attained by naïve Bayes. Similarly, in the case of cross-validation, C4.5 attains (0.9611,0.9817) accuracy and naïve Bayes (0.9096, 0.9420).

It is interesting to note that both algorithms prove to be quite robust when subjected to training data with an increasing amount of label errors. In the case of the test data set, C4.5 maintained a surprisingly high level of accuracy (close to 90%) with 45% of errors in the data, dropping abruptly beyond this level. Naive Bayes went from an average 91.9% accuracy on clean data down to 78% with 50% of errors in the data, but showed a steady 73% of accuracy even with 70% of dirty training data.

	naive Bayes						C4.5					
	test data set			10-fold xval			test data set			10-fold xval		
% errors in labels	sample mean	+/- (*)	Kappa	sample mean	+/- (*)	Kappa	sample mean	+/- (*)	Kappa	sample mean	+/- (*)	Kappa
0	0.9190	0.0169	0.8771	0.9258	0.0162	0.8886	0.9750	0.0097	0.9613	0.9714	0.0103	0.9564
1	0.9160	0.0172	0.8726	0.9141	0.0174	0.8714	0.9740	0.0099	0.9598	0.9604	0.0121	0.9398
5	0.9110	0.0176	0.8652	0.8751	0.0205	0.8149	0.9740	0.0099	0.9598	0.9231	0.0165	0.8839
10	0.8760	0.0204	0.8087	0.8082	0.0244	0.7162	0.9710	0.0104	0.9550	0.8721	0.0207	0.8070
15	0.8490	0.0222	0.7635	0.7377	0.0273	0.6116	0.9720	0.0102	0.9566	0.8257	0.0235	0.7390
20	0.8330	0.0231	0.7380	0.6836	0.0288	0.5380	0.9710	0.0104	0.9550	0.7786	0.0257	0.6692
25	0.8060	0.0245	0.6900	0.6315	0.0299	0.4696	0.9670	0.0111	0.9486	0.7270	0.0276	0.5953
30	0.7960	0.0250	0.6738	0.5848	0.0305	0.4145	0.9620	0.0119	0.9406	0.6797	0.0289	0.5267
35	0.7750	0.0259	0.6386	0.5394	0.0309	0.3597	0.9380	0.0149	0.9025	0.6263	0.0300	0.4488
40	0.7780	0.0258	0.6442	0.5073	0.0310	0.3240	0.9120	0.0176	0.8595	0.5830	0.0306	0.3861
45	0.7810	0.0256	0.6480	0.4761	0.0310	0.2958	0.8900	0.0194	0.8276	0.5306	0.0309	0.2988
50	0.7830	0.0255	0.6509	0.4340	0.0307	0.2549	0.3280	0.0291	0.1544	0.4854	0.0310	0.1333
60	0.7710	0.0260	0.6301	0.3606	0.0298	0.1922	0.1370	0.0213	0.0000	0.5589	0.0308	0.0751
70	0.7310	0.0275	0.5675	0.2994	0.0284	0.1378	0.0510	0.0136	0.0000	0.6368	0.0298	0.1902

(*) confidence: 95%, test sample size: 1000

Table 4: Accuracy of naive Bayes and C4.5 decision tree learner

In the case of cross-validation the performance of both classifiers deteriorated much faster, probably due to the fact that the label errors affect not only the training of the models but also the testing procedure on each of the ten holdouts, during the cross-validation process. The holdouts contain errors themselves on their labels as they form part of the training data set which has been perturbed. This means that there is no golden standard that can be used for comparison purposes: a test deemed as erroneous may be successful and vice versa, due to the nature of the test data at hand. The cross-validation procedure is probably, in this case, a pessimistic measure of the classifier's robustness when dealing with dirty data.
The Kappa statistic is consistent with the accuracy results: with higher levels of agreement it remains closer to 1, moving towards 0 as the quality of the training data decreases.

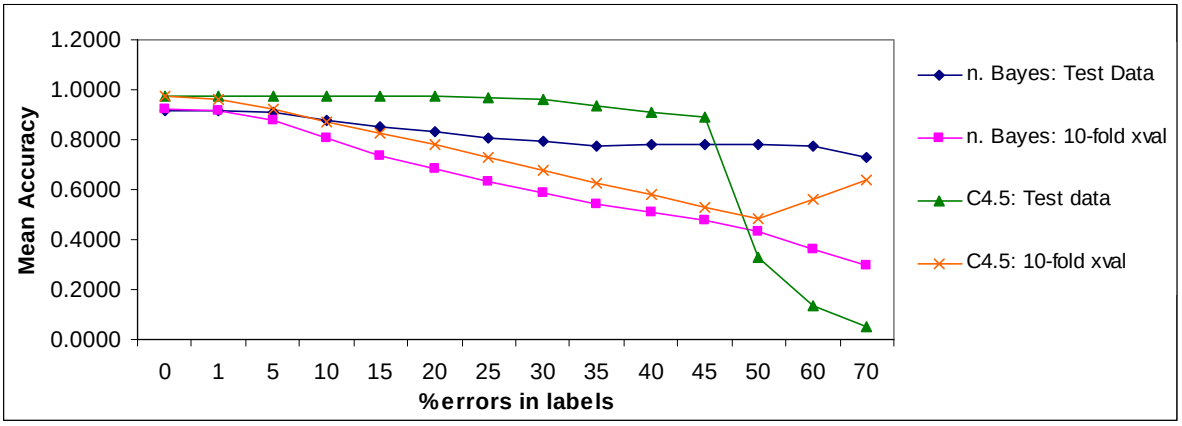

Figure 1: Mean Accuracy as a function of the percentage of dirty training data records

Figure 2 shows that the mean squared error of the probability estimates increases steadily as the quality of the training data is deteriorated. For the C.4.5 classifier, the error starts and 0.039 and increases by a factor of 4 at a label error level of 50%. In the case of naïve Bayes, the mean square error moves from 0.08 with clean data to 0.2382 with 70% errors under cross-validation; and to 0.1440 with 70% errors

using the test data set. In the case of the C4.5 using cross-validation, the reduction in the mean squared error of the probability estimates after reaching a label error level of 50%, which coincides with the increase of accuracy in the classifier is somewhat misleading, but can be explained by assuming that with such error rate, the training samples that were considered normal connections are now labeled as abnormal and vice-versa. As they are also used as testing samples as part of the cross-validation procedure, the overall accuracy seems to be improved and the mean squared error seems to decrease. This is not really so: it is just the effect of using poor quality data for both training and testing purposes.

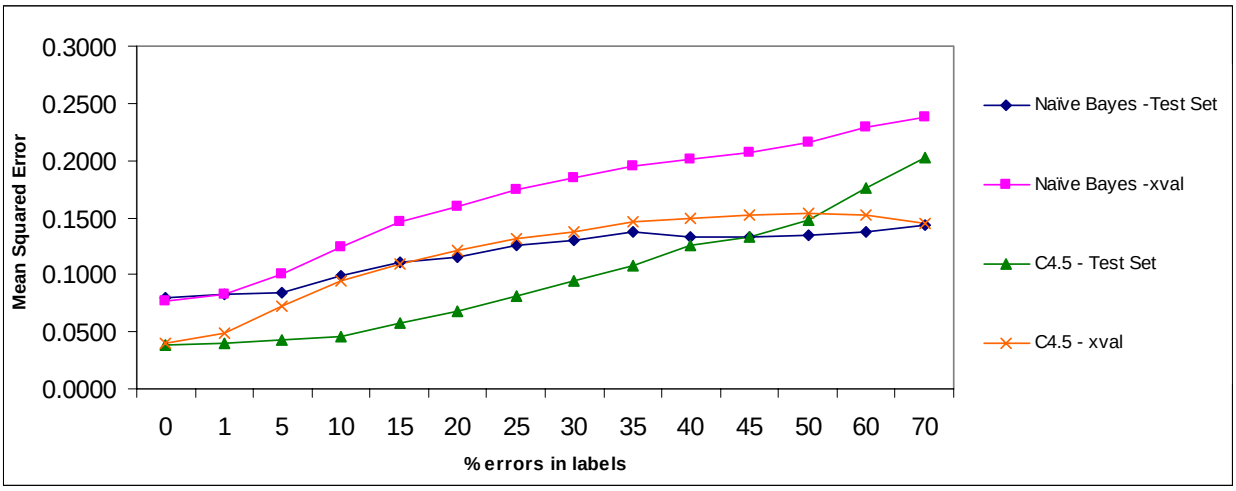

Figure 2: Mean Squared Error

CONCLUSIONS AND FUTURE AREAS OF RESEARCH

Data with totally clean labels may not be required to train a classifier that performs at an acceptable level as a detector of network intrusions. By analyzing the classifiers' performance and determining critical thresholds in the acceptability of errors, boundaries can be established at which point the derived predictions can no longer be trusted.

Our results suggest that even with substantial percentages of error in the labeling of training data, the robustness of classifiers such as C4.5 or naïve Bayes can render high levels of accuracy in network intrusion detection. This means that the trade-off between the quality of the training data and the cost of attaining such quality may be improved if adequate data mining tools are put in place.

In future work, we intend to expand our research to include additional data mining techniques applied to the task of network intrusion detection. We plan to investigate different quality metrics beyond the accuracy dimension considered in this paper and, try to typify different kinds of errors that may arise in different domains.

REFERENCES

[1] Bloedorn E, Christiansen A., Hill W, Skorupka C., Talbot L., Tivel J., "Data Mining for Network Intrusion Detection: How to Get Started", The MITRE Corporation, Technical papers, 2001

[2] Fan W., Lee W., Stolfo S., Miller M., "A Multiple Model Cost-Sensitive Approach for Intrusion Detection". ECML 2000: 142-153

[3] Foody, G., "On the Compensation for Chance Agreement in Image Classification Accuracy Assessment". Photogrammetric Engineering & Remote Sensing. Vol. 58. No. 10. 1992. pp. 1459-1460.

[4] Han J, Kamber M, "*Data Mining Techniques*", Morgan Kaufmann Publishers, 2000

[5] Heady R., Luger G., Maccabe A., and Servilla M., "The architecture of a network level intrusion detection system: Technical report**"**, Computer Science Department, University of New Mexico. 1990

[6] Kohavi R., Sahami M., "Error-Based and Entropy-Based Discretization of Continuous Features", In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 1996. pp. 114-119

[7] Lauria E, Tayi G. *"*Bayesian Data Mining and Knowledge Discovery*"*, in *Data Mining: Opportunities and Challenges*, John Wang (Ed.) Idea Group Publishing, Hershey PA, 2003. pp. 260-277.

[8] Lee W. , Stolfo S., Chan P., Reeves D. "A Data Mining Approach for Building Cost-sensitive and Light Intrusion Detection Models", http://www.cc.gatech.edu/~wenke/project/id.html

[9] Lee W., Stolfo S. "Data Mining Approaches for Intrusion Detection", In *Proceedings of the Seventh USENIX Security Symposium (SECURITY '98)*, San Antonio, TX, 1998

[10] Mitchell T, "*Machine Learning*", McGraw-Hill, 1997

[11] Portnoy L., Eskin E., Stolfo S "Intrusion detection with unlabeled data using clustering". To Appear in *Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001)*. Philadelphia, PA: November 5-8, 2001.

[12] Quinlan, J.R. Simplifying decision trees", International Journal of Man-Machine Studies, 27, 1987. pp. 221-234

[13] Quinlan,J.R.,"*C4.5: Programs for Machine Learning*", Morgan Kauffman, 1993

[14] Strong, D. M., Y. W. Lee, et al. (1997). "Data Quality in Context". Communications of the ACM 40(5) pp. 103-110.

[15] Tayi, G. K. and D. Ballou (1999). "Examining Data Quality". Communications of the ACM 41(2) pp. 54-57

[16] Weka's Website: http://www.cs.waikato.ac.nz/ml/weka/index.html

[17] Witten I., Frank E., "Data Mining: practical machine learning tools and techniques with Java implmentenations**",** Morgan Kaufmann Publishers, 2000