Monitoring Wireless Networks Through Machine Learning Algorithms

Miguel Madeira Teixeira

Thesis to obtain the Master of Science Degree in Telecommunications and Informatics Engineering

Supervisors: Prof. Fernando Henrique Côrte-Real Mira da Silva

Examination Committee

Chairperson: Prof. Paulo Jorge Pires Ferreira
Supervisor: Prof. Fernando Henrique Côrte-Real Mira da Silva
Members of the Committee: Prof. Rui Jorge Morais Tomaz Valadas

November 2015
Acknowledgments

I’d like to thank Professor Fernando Mira da Silva for all the insightful guidance and encouraging words, and without whom this work would not exist. His contribution was invaluable.

I also want to thank Beto and Clarinda (who are my parents) for always supporting me, not only when I had a dissertation to make, but all through my life. I’d like to thank Gonçalo (who is my brother) for always helping me and, when needed, making fun of me, which is another way of helping. And although not present in these years, I’ll always be thankful to Avô Madeira for taking care of me and in the process showing me what life is all about.

Finally I must also thank all the Professors and friends I met during my studying years at Instituto Superior Técnico, whose experiences they provided made me grow not only academically, but as a person. Ivo, Tiago and Gonçalo Nuno deserve a special mention for their participation in this journey.

Thank you all.
Abstract

Detecting anomalies that degrade the user experience in wireless networks is still a process that usually can take too much time or require additional infrastructure. This Dissertation focuses on detecting these problems by using machine learning techniques to create a working prototype, capable of detecting said anomalies as they occur, using exclusively already usually available or easily extracted variables in infrastructure monitoring. For this purpose, anomalies were created, in order to create sets of observation variables corresponding to atypical and potentially anomalous conditions of the network. This data was later pre-processed and used to train a machine learning model, which had its results measured in the form of various performance metrics. A simulation of network usage was made to obtain a ground-truth scenario of the network in a normal state. Data reduction was achieved by using several techniques, and extensive testing was made to attain the best performing conditions. It was also taken into account a possible change in the data over long periods of time, and two ways to cope with it are proposed.

Keywords

Anomaly detection, wireless networks, machine learning, SNMP data.
Resumo

Detetar anomalias que degradem a experiência do utilizador de redes sem fios é um processo normalmente demorado e que requer infraestrutura adicional. Esta Dissertação foca-se na detecção destes problemas usando técnicas de aprendizagem automática, de modo a criar um protótipo funcional capaz de detetar estes eventos à medida que ocorram, usando apenas variáveis já habitualmente disponíveis ou facilmente extraídas na monitorização de infraestruturas. Para este fim, foram criadas anomalias de modo a gerar conjuntos de variáveis de observação correspondente a situações atípicas e potencialmente anômalas da rede. Estes dados foram mais tarde pre-processados e usados para treinar um modelo de aprendizagem automática, cujos resultados foram aferidos através de várias métricas de performance. Uma simulação de uso de rede foi realizada para obter um cenário de uma rede em estado normal como valor de referência. Várias técnicas foram usadas para reduzir a dimensionalidade dos dados, e testes exaustivos foram realizados para se atingirem os melhores resultados. Foi também tida em conta a possibilidade de mudança dos dados com o decorrer de longos períodos de tempo, e são propostas duas maneiras de lidar com a mesma.

Palavras Chave

Deteção de anomalias, redes sem fios, aprendizagem automática, dados SNMP.
# Contents

1 Introduction .................................................. 1
   1.1 Goals ................................................. 3

2 Related Work .................................................. 5
   2.1 Learning and Generalization .............................. 7
      2.1.1 Supervised Learning ........................... 7
      2.1.2 Unsupervised Learning ....................... 10
      2.1.3 Semi-supervised Learning ..................... 11
   2.2 Machine Learning Techniques ............................ 11
      2.2.1 Supervised ..................................... 11
      2.2.2 Unsupervised ................................. 18
   2.3 Machine learning and anomaly detection ............... 22
      2.3.1 Fixed-Background EM Algorithm for Semi-Supervised Anomaly Detection .... 24
      2.3.2 Diagnosing Network-Wide Traffic Anomalies using PCA ....................... 25
   2.4 Feature Selection ...................................... 27
   2.5 Monitoring Wireless Networks ........................... 28
      2.5.1 Simple Network Management Protocol .......... 28
   2.6 Summary ............................................... 29

3 Requirements and Solution .................................. 31
   3.1 Approach .............................................. 33
   3.2 Requirements .......................................... 34
   3.3 Architecture .......................................... 34
   3.4 Solution ............................................... 36
      3.4.1 Data Collection ................................ 36
      3.4.2 Data Pre-processing ............................ 37
      3.4.3 Model Creation and Performance Measurement .......... 37
      3.4.4 Network State Identification ................... 39
4 Data Collection
4.1 Obtaining the ground-truth ..................................................... 43
4.2 Collected Data ......................................................................... 45
4.3 Feature Selection ...................................................................... 46
4.4 Creating the Artificial Anomalies ............................................. 48
5 Experimental Assessment
5.1 Time to train and execute .......................................................... 53
5.2 Big data set results ................................................................. 54
5.3 Small data set results with no model retraining ....................... 57
5.4 Small data set results using retrained models ......................... 63
6 Conclusions
6.1 Achievements ........................................................................... 67
6.2 System Limitations and Future Work ...................................... 67
A SNMP Parameters ..................................................................... 75
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Examples of handwritten digits [1].</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>The process of supervised ML (adapted from [2].)</td>
<td>9</td>
</tr>
<tr>
<td>2.3</td>
<td>ROC graph, with the AUC of the blue classifier represented in grey.</td>
<td>10</td>
</tr>
<tr>
<td>2.4</td>
<td>Optimal classification hyperplane for a linearly separated example, represented as a line. Its exact location depends on two support vectors, seen at distance $d_{\text{max}}$ [3].</td>
<td>12</td>
</tr>
<tr>
<td>2.5</td>
<td>Architecture of an Artificial Neural Network.</td>
<td>16</td>
</tr>
<tr>
<td>2.6</td>
<td>Sigmoid function values as a function of z.</td>
<td>16</td>
</tr>
<tr>
<td>3.1</td>
<td>Diagram of the implemented system.</td>
<td>35</td>
</tr>
<tr>
<td>4.1</td>
<td>Traffic distribution at IP level protocols for an university network.</td>
<td>44</td>
</tr>
<tr>
<td>4.2</td>
<td>Diagram of setup for the data collection.</td>
<td>45</td>
</tr>
<tr>
<td>4.3</td>
<td>Percentage of feature scoring of each SNMP parameter, in descending order.</td>
<td>48</td>
</tr>
<tr>
<td>5.1</td>
<td>ROC graph for the aluminum anomaly with the NN model.</td>
<td>54</td>
</tr>
<tr>
<td>5.2</td>
<td>ROC graph for all anomalies with the NN model.</td>
<td>55</td>
</tr>
<tr>
<td>5.3</td>
<td>ROC graph for the aluminum anomaly with the SVM model.</td>
<td>55</td>
</tr>
<tr>
<td>5.4</td>
<td>ROC graph for all anomalies with the SVM model.</td>
<td>56</td>
</tr>
<tr>
<td>5.5</td>
<td>Feature importance using extra-trees.</td>
<td>58</td>
</tr>
<tr>
<td>5.6</td>
<td>ROC graph achieved with the SVM model and the selected 35 parameters.</td>
<td>59</td>
</tr>
<tr>
<td>5.7</td>
<td>ROC graph achieved with the NN model and the selected 35 parameters.</td>
<td>60</td>
</tr>
<tr>
<td>5.8</td>
<td>Scree plot of the percent variability held by each principal component.</td>
<td>60</td>
</tr>
<tr>
<td>5.9</td>
<td>ROC graph for the SVM model and the small data transformed using 13 PCs.</td>
<td>61</td>
</tr>
<tr>
<td>5.10</td>
<td>AUC achieved by each SVM model, by number of PCs used in the data set.</td>
<td>62</td>
</tr>
<tr>
<td>5.11</td>
<td>ROC graph for the second data set, with the SVM model.</td>
<td>63</td>
</tr>
<tr>
<td>5.12</td>
<td>ROC graph for the second data set, with the NN model.</td>
<td>64</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Results from actual anomalies diagnosed at 99.9% confidence level. 26
2.2 Results from actual anomalies diagnosed at 99.9% confidence level. 26
2.3 Summary of analyzed techniques. 30

4.1 List of applications and their traffic proportions. 45
4.2 The 12 top scoring features using the extra-trees algorithm. 47
4.3 Added features for the data set. 48

5.1 Average time that each ML model took to identify the network status. 53
5.2 Average training time for each ML model. 53
5.3 Scores for the aluminum anomaly with the NN model. 54
5.4 Scores for the aggregation of anomalies with the NN model. 56
5.5 Scores for the aluminum anomaly with the Supervised SVM model. 56
5.6 Scores for the all the anomalies with the Supervised SVM model. 56
5.7 Scores for the all the anomalies with the One-class SVM model. 57
5.8 The 6 top scoring features using the extra-trees algorithm. 59
5.9 Scores for the SVM model, with data transformed using 13 principal components. 62
5.10 Scores for the SVM model with the second data set. 63
5.11 Scores for the NN model with the second data set. 64

A.1 List of the collected SNMP parameters, being the ones included in the final set in bold. 75
A.2 List of the collected SNMP parameters which derived their values from the clients. 81
Acronyms

AP Access Point
AUC Area Under the Curve
CPU Central Processing Unit
CRC Cyclic Redundancy Check
CTS Clear to Send
DoS Denial of Service
EM Expectation Maximization
ESP Encapsulating Security Payload
FANN Fast Artificial Neural Network
FN False Negatives
FP False Positives
FTP File Transfer Protocol
GMM Gaussian Mixture Model
GRE Generic Routing Encapsulation
HTTP Hypertext Transfer Protocol
HTTPS Hypertext Transfer Protocol Secure
IEEE Institute of Electrical and Electronics Engineers
IEFT Internet Engineering Task Force
kNN k-Nearest Neighbor
LOF Local Outlier Factor
MAC Media Access Control
MIB Management Information Base
MIC Message Integrity Check
ML Machine Learning
MPDU MAC Protocol Data Unit
NN Neural Networks
OD Origin-Destination
OID  Object Identifier
OS  Operating Systems
PC  Principal Component
PCA  Principal Component Analysis
P2P  Peer-to-Peer
QoE  Quality of Experience
QoS  Quality of Service
RAM  Random-Access Memory
RBF  Radial Basis Function
RFC  Request for Comments
ROC  Receiver Operating Characteristic
RTS  Request To Send
SNMP  Simple Network Management Protocol
SPE  Squared Prediction Error
SVM  Support Vector Machine
TCP  Transmission Control Protocol
TKIP  Temporal Key Integrity Protocol
TN  True Negatives
TP  True Positives
UDP  User Datagram Protocol
WEP  Wired Equivalent Privacy
1

Introduction

Contents

1.1 Goals .......................................................... 3
Wireless networks are today widely deployed, and often suffer from anomalous events that degrade their quality. These networks are critical for many infrastructures in today's world, making its efficient operation essential [4].

A lot of research goes into discovering internet-related anomalies resulting from malicious attacks [5]. These malicious attacks can be discovered through signature-based methods that encounter widely known anomalies such as Denial of Service (DoS) through their previously known signature. The great majority of techniques is focused on intrusion detection, concerned with guaranteeing network security. Popular network monitoring tools like PRTG Network Monitor\(^1\), that combine several protocols like Cisco's Netflow\(^2\) and Simple Network Management Protocol [6], have the ability to generate an alarm when a certain threshold is passed.

Still, as with signature-based approaches, these methods are not flexible enough to handle new anomalies. More flexible systems are needed to rapidly detect previously unseen anomalies that may not be necessarily related to network security. Some of these anomalies may result from radio problems, interference signals, saturated environments or simply from data transmission errors due to other problems. Machine learning (ML) techniques try to find patterns in data that do not conform to the expected normal behavior, enabling the detection of previously unseen events and thus detecting network anomalies. Also, the problem of discovering other anomalies that can still diminish the Quality of Service (QoS) or Quality of Experience (QoE) of a network, such as radio anomalies, has not been thoroughly addressed in this field.

The objective of this work is to provide a system that can detect these anomalies in an almost real-time through machine learning techniques, requiring no additional infrastructure to the network.

This dissertation is organized as follows. In this first Chapter the goals of this work are described. In Chapter 2 an overview of the related work and the state of the art in this area is presented. Chapter 3 briefly describes the proposed architecture of the approach to the problem, and the implemented solution. Chapter 4 describes the process done to obtain the proper data collection. The tests results analysis and conclusions of this report are presented in Chapter 5 and Chapter 6, respectively.

1.1 Goals

This work addresses the problem of detecting anomalies that degrade user experience in a wireless network, based on machine learning systems.

The goal of this work is to develop models which can do this requiring no additional infrastructure, using only objective and measurable parameters taken from access points (APs). Anomalies originating from both the access points and from external sources are expected to be discovered. The identification

\(^1\)http://www.paessler.com/prtg

\(^2\)http://www.cisco.com/warp/public/732/Tech/netflow
of anomaly situations will be done in a supervised way, using multi-class and one-class classifiers, which are further explained in Chapter 2. For the first one, labeled data consisting of mensurable parameters corresponding to both “non-anomalous” network conditions and anomalous ones will be used, the latter consisting in various injected artificial anomalies which are expected to approximate the conditions found with real anomalies. For the one-class methods only non-anomalous data will be used, and the algorithm will try to find deviations from the normal situation.

Therefore the proposed solution should have the following requirements:

- It should collect and pre-process mensurable network parameters;
- Said network parameters must be obtainable through no additional devices than those already used in the network infrastructure;
- It should detect known and possibly unknown anomaly situations in a wireless network that are affecting the quality of service;
- It should do all the above in almost real-time.

This work is expected to produce the following results: i) specification of the algorithms for detecting the anomalous situation in a timely manner; ii) implementation of a prototype of this system; iii) extensive experimental evaluation with the prototype.
## Related Work

### Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Learning and Generalization</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Machine Learning Techniques</td>
<td>11</td>
</tr>
<tr>
<td>2.3 Machine learning and anomaly detection</td>
<td>22</td>
</tr>
<tr>
<td>2.4 Feature Selection</td>
<td>27</td>
</tr>
<tr>
<td>2.5 Monitoring Wireless Networks</td>
<td>28</td>
</tr>
<tr>
<td>2.6 Summary</td>
<td>29</td>
</tr>
</tbody>
</table>
2.1 Learning and Generalization

Learning automatically from data has played a key role in many areas of science, finance, industry and others. This can be achieved by searching for patterns in data through computer algorithms and discovering regularities which can be used to classify the data into categories, for example. Current applications of algorithms that learn from data include the price prediction of stocks, identification of spam email messages, speech recognition, image recognition, face recognition, search engines and many others.

In a typical scenario, the goal is to predict an outcome measurement, either quantitative (e.g., a stock price) or categorical (e.g., email message is spam/not spam), based on a set of features (such as message length and number of typos in each email). The data from which these features are measured and its outcome observed is called the training set [7], which contains a set of objects (such as stocks or emails). The training set is used to construct a range of prediction models, which are determined during the training or learning phase and will be able to predict the outcome of new unseen objects. The best performing of these models is then chosen by comparing between them using a validation set, which is simply another set of examples, different from the training set. Finally, to evaluate the accuracy of the obtained model, a test set is used, which is another selection of examples, different from the training and validation sets [1].

Defining useful features that are also fast to compute is a crucial step in machine learning, and often done by experts in the domain to which the problem belongs. Besides this, features may be independently not relevant, but become important when combined with other features.

In most of the cases, the original input data must be pre-processed, so that it is transformed into some new space where the pattern recognition problem is expected to be easier to solve. For example, in the speech recognition problem, the values for the length of the words in seconds can be scaled so that every value become a real number between 0 and 1. This would reduce the variability within each word class, which can make it much easier for certain algorithms to learn. This stage can also consist in feature removal, derivation between features, and other techniques. This process can reduce computational complexity, increase the accuracy of the detection algorithm, facilitate data understanding and improve generalization [5]. All the pre-processing applied to the training set must be applied in the same way to the testing and validation sets.

Machine learning techniques may be supervised, semi-supervised or unsupervised.

2.1.1 Supervised Learning

Learning is called supervised if the training data of the application is fully labeled, with the input vectors having corresponding target vectors to guide the learning process. Supervised learning may
be divided in classification and regression problems. When the goal is to assign each input to a finite number of discrete categories, it is a classification problem. If the desired output is of one or more continuous variables, then it consists in a regression task.

As an example, let us consider the problem of recognizing handwritten digits, illustrated in Figure 2.1. The first step is to identify the required data, which in this case would be the digit representation. Here, each digit corresponds to a 28 × 28 pixel image and so can be represented by a vector \( x \) comprising 784 real numbers. Each number corresponds to a feature. Some digits may be unintelligible or not completely scanned, which would translate into noise and missing feature values. These problems are addressed by pre-processing the data, such as simply removing unwanted examples. Then the data is divided into training, test and validation sets. A technique such as those reviewed in Section 2.2.2 is chosen so that it will take vector \( x \) as input and that will classify the output as the correct corresponding digit 0,...,9, after proper training and validation with the training and validation sets, respectively. After the model with the best predictive values is chosen, it’s prediction accuracy is finally measured using the test set. If the results are unsatisfactory it is necessary to return to a previous stage of the supervised machine learning process (as seen in Figure 2.2). The possible reasons may be that relevant features are not being used, the number of features (also called dimensionality) may be too high, the selected technique is inappropriate, finer parameter tuning is needed, a larger training set is required or the classes representation in the data set is too imbalanced [2].

There are various methods for evaluating a supervised learning model. For this work, the most relevant is the one used for binary classification problems. As an example let us consider the task of classifying an event as anomalous (positive) or non-anomalous (negative). When the model correctly predicts a positive or negative result, it is called a true positive (TP) or true negative (TN), respectively. If an example is classified incorrectly it will be called either a false positive (FP) or a false negative (FN), accordingly. Popular metrics for binary classifiers include the recall, which is the fraction of relevant instances that are retrieved, given by [8]:

\[
Recall = \frac{TP}{TP + FN}.
\]  

If a classifier achieves a value of 1.0 in recall score for a certain class it means that every item from that class was correctly labeled, but gives no information about how many items were incorrectly also
labeled as being from that class. For the latter information there exists another metric, the precision, which is the fraction of retrieved instances that are relevant, given by:

\[ \text{Precision} = \frac{TP}{TP + FP} \]  \hspace{1cm} (2.2)

These two metrics are usually discussed combined. The F1 score, given by:

\[ F1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \]  \hspace{1cm} (2.3)

is one of these combinations. It can be interpreted as the weighted average of the precision and the recall.

Another metric is given by the accuracy score, defined by:

\[ \text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \]  \hspace{1cm} (2.4)

As showed, accuracy can be seen as the proportion of true results in all the results.

Another technique is the Area Under the Receiver Operating Characteristic Curve (ROC), or simply
Area Under the Curve (AUC), useful for classifiers that output a probability value. In a binary classification problem, a certain threshold must be set to decide when the example belongs to one class or the other. Usually as the threshold is decreased, more true and false positives are obtained. A Receiver Operating Characteristic graph is a two-dimensional graph in which the true positives rate is plotted on the Y axis, and the false positives rate is plotted on the X axis. For a certain classifier, a different value for the threshold will then result in a different point in the ROC graph. A curve can be drawn by varying the threshold from 0 to 1. AUC summarizes the ROC performance to a single scalar value representing expected performance. Since it is a portion of the area of the unit square, its value will always be between 0 (failing all predictions) and 1.0 (guessing correctly all predictions). Since random guessing produces the diagonal line between (0,0) and (1,1), (i.e., an AUC of 0.5), as seen in Figure 2.3, no realistic classifier should have a AUC less than 0.5. It has the advantage of being insensitive to imbalanced classes, while accuracy for example is not [9].

![ROC graph, with the AUC of the blue classifier represented in grey.](image)

**Figure 2.3:** ROC graph, with the AUC of the blue classifier represented in grey.

### 2.1.2 Unsupervised Learning

If the training data consists of a set of only input vectors $x$ without any corresponding target values (unlabeled data), the problem is one of unsupervised learning. Commonly the goal in these problems is to discover groups of similar examples within the data, which is known as *clustering*, or to determine the distribution of data within the input space, called *density estimation*, or to project the data from a high-dimensional space down to three or two dimensions for the purpose of *visualization* or *dimensionality reduction* [1] [10] [8].

One example may be the case where the data set consists of people’s sizes and the goal is to determine the most useful t-shirt sizes. There is no target value for the size of each person because the objective is to create the minimum amount of t-shirt sizes that will fit the most people. In this example one would have to decide how many clusters of sizes would be ideal before training the model, since...
as there are no target values the algorithm itself is able to provide many different answers. Another example is face clustering in a set of photos, as used in the Facenet system [11] for Google Photos.

For clustering problems, any evaluation metric should try to answer if the clustering obtained defines separations of the data similar to some ground truth set of classes, or if it satisfies some assumption such that members belonging to the same class are more similar between themselves than to members of different classes, according to some similarity metric\(^1\). For other problems other evaluations are required. The evaluation for unsupervised learning techniques thus depends on the problem it is trying to solve, since its result must be interpreted.

### 2.1.3 Semi-supervised Learning

When both labeled and unlabeled data is used for training, the learning is called semi-supervised. The objective is to generate a model that achieves better results than if it used only labeled data [7]. For this reason it is generally classified as a subclass of supervised learning, when labeled data is relatively small compared to the unlabeled data, and is specially useful when the cost associated with the labeling process is prohibitive, but the acquisition of unlabeled data is relatively inexpensive.

The evaluation of the produced model can be done in the same way one would to a supervised model.

### 2.2 Machine Learning Techniques

#### 2.2.1 Supervised

There are many supervised learning techniques, and in this section, some of the most relevant ones in the context of this work will be briefly reviewed.

**Support Vector Machines** A support vector machine (SVM) is a technique used for linear and non-linear data classification and regression analysis. For classification, given a set of labeled training examples, each belonging to one of two categories, it produces a model which predicts the correct category of new examples. This model is a representation of the training data as points into a higher (possibly infinite) dimensional space, mapped so that the examples of each category are separated by a hyperplane with a margin as wide as possible [12].

Let’s start with the linear case and assume a training set of instance-label pairs \((x_i, y_i), i = 1, \ldots, n\) where \(x_i \in \mathbb{R}^N\) and \(y \in \{1, -1\}^n\). Being \(x\) a vector with components \(x_i\), \(w\) a weight vector, and bias \(b\), the discriminant function will be of the form

\[w^T x + b = 0\]
Figure 2.4: Optimal classification hyperplane for a linearly separated example, represented as a line. Its exact location depends on two support vectors, seen at distance \(d_{\text{max}}\).

\[
f(x) = w^T x + b.
\]  

(2.5)

where \(w^T x\) is the dot product between the weight \(w\) and the examples \(x\), i.e., \(\sum_i w_i x_i\). The set of points \(x\) such that

\[
w^T x = -b
\]  

(2.6)

will result in a line, a plane or an hyperplane, which will be the decision boundary of the classifier. The bias \(b\) translates the hyperplane away from the origin, so in the case \(b = 0\) the hyperplane goes through the origin. As shown in Figure 2.4, the exact location of the hyperplane should only depend on the data patterns laying near to the decision boundary - the support vectors. The sign of the discriminant function \(f(x)\) denotes the side of the hyperplane a point is on, i.e., the class to which it belongs.

The geometric margin we want to maximize is given by \(1/\|w\|^2\), which is equivalent to minimizing \(\|w\|^2\). Since the data may not be linearly separable the classifier must be allowed to misclassify some points, or to allow some points in the margin. To this effect, slack variables are added, in the form of \(\xi_i \geq 0\), making the SVM more robust to outliers. Due to this, the objective of minimizing will be augmented with a term \(C \sum_i \xi_i\) to penalize the classification errors. The constant \(C > 0\) sets the relative importance of maximizing the margin and minimizing the amount of slack. This leads to the following optimization problem [13]:

minimize \[ \frac{1}{2} \| w \|^2 + C \sum_{i=1}^{n} \xi_i \] subject to: \[ y_i (w^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0. \] (2.7)

The constraint in this formulation ensures that the maximum margin classifier classifies each example correctly.

Contrarily to the previous example, many problems cannot be solved linearly. With SVMs it is possible to make a non-linear problem in a linear one in a higher dimensional feature space. The data is mapped from the input space \( N \) to a feature space \( F \), using a non-linear function \( \phi : N \mapsto F \). In the space \( F \) the discriminant function is:

\[ f(x) = w^T \phi(x) + b. \] (2.8)

A kernel is a similarity function much easier to compute than its corresponding feature vectors, and which under certain constraints can be expressed as a dot product in a possibly infinite dimensional feature space. Depending on the relationship between features, different kernels may be required. The linear kernel is not sufficient for non-linear relationships between features, where maybe the Gaussian, or radial basis function (RBF), kernel may be used instead. Representing points in the high-dimensional space explicitly could lead to much processing time, but SVMs avoid this and achieve a lower computational cost by using the “Kernel Trick” [14], which consists in using a kernel function to compute the dot products.

Since SVMs depend on the data only through dot products [15], kernels can be computed instead of feature vectors, thus enabling relatively good performance for this technique.

The kernels may depend on its own parameters, and careful tuning must be done to avoid problems such as overfitting. Overfitting consists in the achieved model adjusting itself to irrelevant relationships between features or random error in a given data set, producing great results in that set but tipically bad results in other slightly different sets. It can be seen as a model learning how to excel for a given data set at the expense of generalization. Because of this, a validation set is required to find the best kernel parameters and constant \( C \). This is typically done through a grid search. Grid searching is the process of finding the best combination of values for the algorithm’s hyperparameters by comparing the results of all possibilities with a validation set. The range of possible values for each hyperparameter is previously defined by the user, with its values being typically chosen on a logarithmic scale. The proper procedure for using an SVM with the Gaussian kernel, for example, consists in the following steps [16]:

1. Pre-process the data and separate it in training, cross-validation and test set, normalizing it properly.
2. Use the validation set to find the best combination of parameters \( \gamma \), which is the width of the
Gaussian kernel, and $C$.

3. Use the obtained parameters to train the whole training set.

4. Finally test the SVM with the test set.

The lack of proper normalization can severely degrade the accuracy of the SVM [17]. As happens when other classifiers, unbalanced data sets where one class is much more represented than the other can be a challenge to SVMs. The most common way to solve this problem is to assign different costs for misclassification to each class. So the total misclassification cost, $C \sum_{i}^{n} \xi_{i}$, is replaced with two terms:

$$C \sum_{i}^{n} \xi_{i} \mapsto C_{+} \sum_{i \in I_{+}} \xi_{i} + C_{-} \sum_{i \in I_{-}} \xi_{i}, \tag{2.9}$$

where $C_{+}$ ($C_{-}$) is the constant for the positive (negative) examples and $I_{+}$ ($I_{-}$) are the sets of positive (negative) examples. In order for the total penalty of each class to be equal, it is defined that:

$$\frac{C_{+}}{C_{-}} = \frac{n_{-}}{n_{+}},$$

where $n_{+}$ and $n_{-}$ is the number of positive and negative examples, respectively [13].

It is usually advised to remove features that do not contribute to the accuracy of a supervised classifier to improve accuracy [18], but that doesn’t seem to have much effect in SVMs. On the other hand, support vector machines do not give much insight on why a particular prediction was made, and reducing the set of features can help in better understanding the data.

There exist two popular ways to do multi-class classification in SVM: one class against the rest of the classes, or one against one. The first consists in constructing $k$ SVM models where $k$ is the number of classes, and the $i$th model is trained with the examples in the $i$th class being the only positive labels. The second way constructs $k(k - 1)/2$ classifiers, where each one is trained on data from two of the classes. Both seem to have comparable performance, but the one-against-one method training time is shorter [19].

A typical variant of the standard SVM is the one-class SVM, typically used in novelty detection as proposed in [20]. Here, only the data pertaining to normal conditions is used to train the SVM, and each example is classified as belonging or not to that class. This differing approach has its performance strongly influenced by its requirement to fix $a priori$ the percentage of non-anomalous data allowed to fall outside the description of the non-anomalous class [21].

SVM training always finds a global minimum, and thus it does not depend on each run. The computational complexity of training linear SVMs is linear in the number of examples, $O(n)$. For non-linear SVMs the computational complexity achieved is between quadratic, $O(n^2)$, and cubic, $O(n^3)$, depending on whether $C$ is small or large, respectively [22]. The prediction time for a new example is $O(n_{SV} \cdot f)$. 

14
time, where \( n_{SV} \) is the number of support vectors and \( f \) is the number of features.

**Artificial Neural Networks**  Artificial Neural Networks, sometimes just called Neural Networks (NN), are a supervised learning technique inspired by how biological brains work, and it is based on a model of neurons. The first artificial neurons were called perceptrons. Perceptrons take several weighted binary inputs, also known as the input layer, and produce a single binary output (the output layer), determined by whether the weighted sum \( \sum m w_m x_m \) is less than or greater than some threshold value. In that sum, \( x_m \) is the binary input of index \( m \) and \( w_m \) its respective weight. Considering that \( \sum m w_m x_m = w \cdot x \) and defining a perceptron’s bias as \( b = -\text{threshold} \), the output is given by a function called the activation function, in this case defined by:

\[
\text{output} = \begin{cases} 
0 & \text{if } w \cdot x + b \leq 0 \\
1 & \text{if } w \cdot x + b > 0 
\end{cases}
\] (2.10)

The minimum square error cost function is typically used to quantify how well the output from the network, \( h_{w,b} \), approximates the known corresponding values for the training inputs \( x \), given below by \( y(x) \):

\[
C(w, b) = \frac{1}{2} \sum_x \|y(x) - h_{w,b}(x)\|^2,
\] (2.11)

By combining the results of several perceptrons, a more complex decision can be made.

If instead of having only one layer of neurons more layers are added which take as input the output of the previous combination of neurons, as seen in Figure 2.5, an even more complex and abstract decision can be reached. Networks with this architecture and using another type of neurons, such as the sigmoid neuron (explained further below) can solve non-linear problems. This process of adding more layers can result in progressively more sophisticated decision-making. All the layers in-between the output and the input layers are called the hidden layers (and its units the hidden units). Networks like this, where the information moves only forward, from the input nodes through the hidden nodes, if any, and to the output nodes, are called feedforward neural networks. By adapting the weights on the incoming connections of these hidden units the network learns feature detectors that enable it to predict the correct output, given an input vector.

To train multi-layered networks, an algorithm called backpropagation is used, which calculates the error at the output nodes, where the formula is simpler, and then propagates them back through the network using gradient descent [23] with respect to the weights and the biases. It requires the cost function to be differentiable, and discovers the set of weights and biases which make the cost as small as possible. Since the activation function of perceptrons networks is not differentiable, another type of neurons must be used, such as the sigmoid neuron. In sigmoid neurons, both the inputs and outputs
are no longer binary, but can take any value between 0 and 1. Its activation function is defined by:

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \quad (2.12)$$

where $z = w_m \cdot x_m + b$. As can be seen in Figure 2.6, when $z$ is large and positive, the output is approximately 1, and when $z$ is large and negative the output is approximately 0.

Since the learning was prohibitively slow on higher layer networks, only recently there have been benefits from adding more than one hidden layer to neural networks, creating the so-called deep neural networks. This slowness was due to the “vanishing effect” problem [24], which consisted in the fact that as the error signal was backward propagated through the layers during backpropagation, the non-linear interaction made it very difficult for the lowest layers to adjust their weights accordingly to the output targeted by the output layer. This happens because the gradients used in backpropagation begin to vanish and become relatively small when compared to the weights.

In 2006 Geoffrey Hinton [25] developed a technique that made these networks much faster to train. By using the first hidden layers to search for regularities in the data, pre-training them in problem-agnostic, unsupervised, greedy manner, it became possible to create input more useful for higher layers.
using the so called Boltzmann restricted machines [26]. On the other hand, these deep neural networks can also lead to overfitting. More recently, the vanishing effect problem has been further diminished, specially in classification problems, by a different approach that consists in using different activation functions, such as the rectifier linear unit [27] and the maxout [28].

The overfitting problem has recently been addressed by a regularization technique called "dropout" [29], in reference to dropping out units in a neural network. Dropping out means that a hidden or visible unit will be removed from the network, along with its incoming and outgoing edges. This prevents complex co-adaptations where the network would learn feature detectors that would only be helpful in the context of several other specific feature detectors. Instead, each neuron learns to detect a feature that is generally helpful for producing the correct answers.

Deep neural networks often achieve better results on a lot of different applications since they can build up a complex hierarchy of concepts that might be useful in deciding the correct output. The additional layers increase the modeling capability of the neural network by adding levels of abstraction. Besides that, for some classification problems deep neural networks do not need hand-engineered features, since they are learned by the network.

There exist other types of artificial neural networks, such as the radial basis function network [30], that has the particularity of using radial basis functions (e.g., Gaussian) as activation functions. They can be used for classification, system control or time series prediction, for example. Another type of neural networks, for example, is the autoencoder, which differs in the fact that it has as many output as input nodes, and it tries to reconstruct its own inputs instead of predicting some target values. It learns a compressed representation for a set of data, typically for the purpose of dimensionality reduction.

Artificial Neural Networks do not necessarily converge to a global minimum error, only a local minimum. This means that each run can, in theory, achieve different results. The runtime is proportional to the number of weights, but their time complexity is hard to define, as it depends on each implementation.

Other supervised techniques

Other supervised techniques exist that weren’t as thoroughly analyzed in this work. k-Nearest Neighbor (kNN) is one of the simplest forms of supervised learning. It is based on the principle that the instances within a data set will generally exist in close proximity, according to certain distance metrics, to other instances that have similar properties. If the instances are tagged with a classification label, then the value of the label of an unclassified instance can be determined by observing the class of its nearest neighbors. The kNN locates the k nearest instances to the query instance and determines its class by identifying the single most frequent class label [2]. It compares new instances with instances seen in training which have been stored in memory instead of performing explicit generalization. As a result it can take too much time to discover which class a new example belongs to, since all the data points collected may be needed to determine it [5]. This condition makes this algorithm less desirable
for the supervised approach of this work, as it would take too much time. Also, the performance of this algorithms suffers greatly in high-dimensions, which is another characteristic of this work.

A commonly used and very simple classifier is the Naive Bayes classifier, which for each given instance outputs a probability distribution for the set of classes. It is based on applying the Bayes theorem with strong independence assumptions among the features. Its accuracy is usually outperformed by other methods such as SVMs and Neural Networks and it is intolerant to highly interdependent attributes [2], making it less interesting for the problem studied in this work. Its advantages include being relatively fast to train and not requiring too much training data.

Logistic regression [31] is another commonly used and simple method that can be used for supervised classification (despite having regression in its name). It estimates a probability for each class an instance can take, modeled as a function of the features using a logistic function, which is a special case of the sigmoid function referenced earlier. This probability is translated into the instance belonging or not to each class using a chosen threshold [1]. It assumes that there is one smooth linear decision boundary, which limits its applicability, but scales relatively well. It is outperformed in various classification tasks by more sophisticated methods such as SVMs and Neural Networks.

Another supervised method is the Random Forests [34], that consists in constructing multiple decision trees (further explained in Section 2.4) in training time to achieve a better predictive performance. Its output for any given instance is the class that is the mode of the outputs from the constructed decision trees. This algorithm also has the advantage of easy interpretation of the prediction it makes, and scaling relatively well.

2.2.2 Unsupervised

There are many unsupervised learning techniques. In this section, some of the most relevant ones in the context of this work will be briefly reviewed.

Principal Component Analysis Principal Component Analysis (PCA) is a coordinate transformation method that maps a given set of \( n \)-dimensional data points onto new axes, called principal axes or principal components (PCs). Principal components are a set of linearly uncorrelated variables, and its number is less than or equal to \( n \). When working with zero-mean data, each principal component has the property that it points in the direction of maximum variance remaining in the data, given the variance already accounted by the preceding components. Each principal component has the restriction that it is orthogonal to (or uncorrelated with) the previous components. Applying PCA to a previously normalized \( m \times n \) data matrix \( X \), where each row is a point in \( \mathbb{R}^m \), the first principal component is given by the vector:
\[ v_1 = \arg \max_{\|v\| = 1} \|Xv\| \]  
\[ (2.13) \]

where \( \|v\| \) is the Euclidian norm of \( v \), and \( \|Xv\| \) is proportional to the data variance measured along the vector \( v \). It follows that the \( i \)th principal component is given by

\[ v_i = \arg \max_{\|v\| = 1} \|X(1 - \sum_{j=1}^{i-1} v_j v_j^T)v\| \]  
\[ (2.14) \]

As such, the first principal component captures the greatest variance of the data possible on a single axis, and each subsequent principal component the maximum variance among the remaining orthogonal directions. Therefore, the principal components are ordered by the amount of variance they capture [35].

This method is commonly used for data dimension reduction. On the assumption that relevant data information is encoded along \( k < n \) largest variance directions, data dimension can be reduced to \( k \).

In [35], it was shown that PCA-based diagnostic scheme outperforms significantly other diagnostic schemes, such as those using signal processing techniques (wavelet/Fourier analysis). On the other hand, there is no direct mapping between PCA’s dimensionality-reduced subspace and the original location of the anomaly [36]. Another relevant problem lies in the fact that large anomalies may deteriorate the performance of PCA for anomaly detection by contaminating the non-anomalous subspace [37]. Still in [36], it was shown that the assumption that the first few principal components capture the vast majority of the variance in the data cannot be generalized to all networks and traffic aggregations, and that the effectiveness of PCA is very sensitive to the the number of components included in the normal subspace. In general, Cattel’s Scree Test [38] is within one or two principal components of the number that minimizes the false-positive rate. Another result found in this study is the importance of pre-processing the data to identify and remove large anomalies before constructing the normal subspace, as their inclusion in it may yield false positives or false negatives.

Variants of the main algorithm exist that address some of these disadvantages. Robust PCA techniques [39] are less sensitive to outliers, presenting more robust estimates at contaminated data. Kernel PCA techniques [40] allow PCA to be performed into a higher-dimensional feature space, achieving better results in some problems.

**k-Means**  

k-Means is a simple clustering algorithm which is used to automatically recognize groups of similar data objects, classifying them to a \( k \) number of clusters previously specified by the user. It first randomly chooses a set of \( k \) data points (not necessarily objects of the data set) as centroids. After that, the (usually Euclidean) distance from each data object to all centroids is calculated, and each one is assigned to the nearest centroid. After this the positions of all the centroids are recalculated, and the process is iterated until the centroids do not change. As a result, \( k \) clusters are found representing a set of \( n \) data objects [41].
It's main weaknesses are: it may terminate at local optimum; it is very sensitive to noisy data and outliers, given that the data distribution may be distorted or biased in the presence of an outlier with a very large value; it is not suitable to discover clusters with non-convex shape or clusters of very different size; it is needed to specify $k$; the result and total run time depend upon the initial partition [42]. In [43] a method using a combination of k-means with distance-based outlier detection outperformed other methods based on support vector machines.

This algorithm is relatively scalable and efficient in processing large data sets, having a complexity of $O(ikn)$, where $i$ is the total number of iterations.

**k-Medoids** The k-Medoids is a similar clustering method that tries to overcome the problem of outlier sensitivity found in k-Means. A medoid may be defined as the object of a cluster whose average dissimilarity to all objects in the cluster is minimal, i.e., the most centrally located data object in a cluster [41] [42].

In this algorithm $k$ medoids, rather than centroids (as in K-means), are picked to represent $k$ clusters. The partitioning method is performed based on the principle of minimizing the sum of dissimilarities between each data object in a cluster, using the following absolute-error criterion [44]:

$$E = \sum_{j=1}^{k} \sum_{p \in C_j} |p - o_j|$$  \hspace{1cm} (2.15)

Where $E$ is the sum of the absolute error for all objects in the data set, $p$ is the point in space representing a given object in cluster $C_j$, and $o_j$ is the representative object of $C_j$. The cost function is the difference in absolute-error value if a current representative object is replaced by a non-representative object.

In the most common k-medoids clustering realization, Partitioning Around Medoids, a previously specified $k$ number of $o_j$ representative objects are arbitrarily chosen from the data set. After this, the following process is repeated until no change is obtained: each remaining object is assigned to the cluster with the nearest $o_j$; for each non-representative object $o_{random}$ associated with each $o_j$, the total cost $S$ of swapping it with $o_j$, i.e., the sum of costs incurred by all non-representative objects, is calculated; if $S < 0$ then $o_j$ is replaced with $o_{random}$, and a new set of $k$ $o_j$ representative objects is formed.

In [45] k-medoids and k-means were each combined with Naïve Bayes classification (a simple supervised learning technique) for anomaly based intrusion detection. The combination with k-medoids clustering achieved better results than the one with k-means. The improvement was shown to increase with the data set size. Nevertheless, the authors recommended the combination of k-medoids with better supervised learning algorithm like Support Vector Machines.
As with k-means, a different initialization can lead to different results, because the result can fall in a local optimum. Also, it tends to form convex clusters and it is difficult to find the optimal number of clusters.

This algorithm is relatively more costly than k-Means, since its complexity is $O(i (n - k)^2)$, where $i$ is the total number of iterations. The algorithm iterates until each representative object is actually the medoid of its cluster. As with k-Means, the result and total run time depend upon the initial partition.

**Expectation Maximization to estimate a Mixture of Gaussians** In the Expectation Maximization (EM) clustering, an EM algorithm is used to find the parameters which maximize the likelihood of the data, assuming it is generated from a linear combination of $k$ multivariate normal distributions, also known as Gaussian distributions. The probability density of a multivariate Gaussian with mean $\mu_m$ and covariance matrix $\Sigma_m$ at $x$ is represented by

$$
\mathcal{N}(x|\mu_m, \Sigma_m)
$$

The mixture of $M$ multivariate Gaussian distributions is defined as

$$
p(x|\theta) = \sum_{m=1}^{M} \pi_m \mathcal{N}(x|\mu_m, \Sigma_m),
$$

(2.16)

where the $\pi_m$ are mixture proportions (or coefficients) which satisfy $\pi_m \geq 0$ and $\sum_{m=1}^{M} \pi_m = 1$, and $\theta = \{\pi_m, \mu_m, \Sigma_m\}$ represents the parameters of the mixture model with $M$ components.

The EM algorithm learns both the means and the covariance of the normal distributions [46]. If missing data is introduced, it can be used to maximize the resulting expected complete data log-likelihood. Its input parameters are the data set, the total number of clusters, the maximum error tolerance and the maximum number of iterations. It consists of two steps, the Expectation Step (E-step) and the Maximization Step (M-step). Starting values are chosen for the model parameters and given as input to the E-step. This first step computes the posterior probabilities for each data point $x_i$ being generated by the $m$th component:

$$
p(z_{im} = 1|z_i, \theta^k) = \frac{\pi_m^k \mathcal{N}(x_i|\mu_m^k, \Sigma_m^k)}{\sum_{m'=1}^{M} \pi_m^k \mathcal{N}(x_i|\mu_m^k, \Sigma_{m'}^k)} \equiv \gamma_{im}^k
$$

(2.17)

Here, $\theta^k$ contains the parameter estimates at the $k$th iteration and $z_i$ indicates the component which generated the $i$th data point. In the M-step, the expected complete data log-likelihood is maximized to produce the updated parameter values, in this case $\pi_m, \mu_m$, and $\Sigma_m$. These are updated according to the following equations:

$$
\pi_{m}^{k+1} = \frac{1}{N} \sum_{i=1}^{N} \gamma_{im}^k,
$$

(2.18)
\[ \mu_{m}^{k+1} = \frac{\sum_{i=1}^{N} \gamma_{im}^{k} x_i}{\sum_{i=1}^{N} \gamma_{im}^{k}} \quad (2.19) \]

\[ \sum_{m=1}^{k+1} = \frac{\sum_{i=1}^{N} \gamma_{im}^{k} (x_i - \mu_{m}^{k+1}) (x_i - \mu_{m}^{k+1})^{T}}{\sum_{i=1}^{N} \gamma_{im}^{k}} \quad (2.20) \]

The outputs of the M-step (that is, the values of the parameters) are then used as inputs for the next E-step. The process is iterated between E-steps and M-steps until convergence is reached \[47\] \[41\]. In \[48\] it is shown that each iteration of the EM algorithm increases the log-likelihood until a local maximum is found.

A Gaussian Mixture Model (GMM) has been shown in \[49\] to be more appropriate than k-means and k-medoids, considering that the clusters have different sizes and are correlated. Also, this technique obtains soft assignment of points to clusters, instead of hard assignment, and does not assume spherical clusters.

Other techniques Other techniques exist that can be applied to anomaly detection, but for a number of reasons were not thoroughly analyzed here.

Nearest neighbor based algorithms like Local Outlier Factor (LOF) \[50\] and Local Outlier Probabilities \[51\] do not rely on generalization, and can take too much time to discover which class a new example belongs to, since all the data points collected may be needed to determine it \[5\]. This condition would make the requirement of a timely detection of an anomaly impractical. Also, the performance of these algorithms suffers greatly in high-dimensions, since the similarity metrics do not consider the possible relations between attributes. Besides this, LOF does not handle well non-uniform clusters \[52\].

A subset of Neural Networks named Replicator Neural Networks was also proposed for outlier discovery \[53\] \[54\], but no further studies were made about its effectiveness and the implementation used is not publicly available.

### 2.3 Machine learning and anomaly detection

Anomaly, outlier or novelty detection consists in identifying observations from a data set which deviate so much from the other observations as to arouse suspicions that it was generated by a different mechanism \[55\]. Anomalies are distinct from unwanted noise in the data, which can be defined as phenomenon which is not of interest to the analyst, but acts as a hindrance to data analysis. Thus, it is needed to define a region in data representing normal behavior and declare any observation in the data which does not belong to this normal region as an anomaly. This requirement comes with associated challenges \[56\]:

- The data will often contain noise which tends to be similar to the actual anomalies, making it difficult
to distinguish;

- In many domains normal behavior keeps evolving and a current notion of normal behavior might not be sufficiently representative in the future;

- Defining a normal region which encompasses every possible normal behavior is very difficult and the boundary between normal and anomalous behavior is often not precise;

- Availability of labeled data for training, validation and test of models used by anomaly detection techniques can be a major issue.

In supervised and semi-supervised learning, the two possibilities are between multi-class and one-class classification approaches. When using multiple classes there is usually the issue of the anomalous instances being far fewer compared to the normal instances (i.e., unbalanced data set) [57]. This problem can be addressed by inserting artificial anomalies in the non-anomalous data set [58], which are expected to approximate the conditions found with real anomalies. Having done this, the problem becomes one of multi-class classification, possibly a binary one if the artificial anomalies are all labeled simply as an unspecific “anomaly”. However, another problem persists: the labeled “normal” data set may be contaminated with unknown anomalies. Also if what constitutes “normal” data changes over time the algorithm will have to be retrained manually. Real anomalies may be very difficult or impossible to detect if they are not similar to the artificially created anomalies. The general advantages are the high accuracy achieved in comparison to other methods and fast classification.

In the one-class approach to anomaly detection two types of data sets may be used, where the first one is labeled and contains the data labeled as non-anomalous, and the second one, used for validation or testing, can contain both normal and abnormal observations [59]. As an advantage new anomalies can be more easily detected, since the data is tested only to being different from the normal class. In this way the previous problem of unbalanced data set is addressed, but the problem of the “normal” data set being possibly contaminated remains. The algorithm has to be manually retrained if what constitutes non-anomalous data changes over time.

In the unsupervised category it is implicitly assumed that normal instances are far more frequent than anomalies in the test data, which not rarely leads to false positives [60]. If this assumption is not true, then such techniques suffer from high false alarm rates. These techniques are often difficult to evaluate, and its parameters difficult to tune optimally. The testing of whether a new example is normal or not may take too long on some algorithms. On the other hand, they can adapt automatically to a change of the definition of normal data, as well as discover new anomalies [61].

Below, two relevant studies using machine learning techniques to detect anomalies will be analyzed.
2.3.1 Fixed-Background EM Algorithm for Semi-Supervised Anomaly Detection

In [59] a study was conducted regarding a semi-supervised anomaly detection problem where anomalies lie in the normal data. Anomalies were identified collectively rather than individually, based on deviations from the distribution of the labeled "normal" data. The method also assumes non-anomalous data has a fixed distribution and that anomalies occur as an excess in the distribution of the normal data.

The labeled data, without anomalies, was first modeled using a mixture of Gaussians. This model will be called $p_N(x)$ and was learned using the EM algorithm. After this a variant of the EM algorithm was used in an unlabeled data set to fit a mixture of that model and a number of additional Gaussians, corresponding to the additional anomaly model, $p_A(x)$, which resulted in a probabilistic model. This probabilistic model is called the fixed-background model, $p_{FB}(x)$, and is defined as:

$$p_{FB}(x) = (1 - \lambda)p_N(x) + \lambda p_A(x),$$  \hfill (2.21)

where $\lambda$ is the proportion of anomalous observations in the model. The goal is to find an optimal combination of the parameters of the anomaly model, in this case $(\mu_A, \sigma_A)$ and the mixing coefficient $\lambda$. The posterior probability is used as a discriminant function $D$ for an event-by-event anomaly detection:

$$p(\text{anomaly}|x) = \frac{\lambda p_A(x)}{(1 - \lambda)p_N(x) + \lambda p_A(x)} = D(x).$$  \hfill (2.22)

A constant $T \in [0,1]$ serves as a threshold which can be used to control the sensitivity of the classifier:

$$D = \begin{cases} \geq T & \Rightarrow x \text{ is an anomaly} \\ < T & \Rightarrow x \text{ is not an anomaly} \end{cases}$$  \hfill (2.23)

The detailed equations of the variant of the EM algorithm are explained in [59]. Here, the normal model $p_N(x)$ will be fixed, and both $\lambda$ and the parameters of $p_A(x)$ need to be optimized to maximize the log-likelihood.

After fitting the fixed-background model $p_{FB}(x)$ to the unlabeled data that potentially contains anomalies, a likelihood ratio test is used to determine if the anomaly model contributes with real anomalous events, by testing the background only null hypothesis $H_0$, i.e., $\lambda = 0$, against the anomaly hypothesis where $\lambda > 0$. A method called nonparametric bootstrap simulation is used to establish if the null hypothesis can be rejected at a certain desired significance level.

The proposed method was tested with both artificial data generated from a mixture of gaussians and data from a simulated signal produced by the Higgs boson. In the artificial data test case, the algorithm trained the fixed-background model, and with certain thresholds the data was classified as either non-anomalous or anomalous. The classifier performance values obtained were practically identical to those obtained by using the original generative model, only suffering when the test data contained less than
3% of anomalies. However, the estimated proportion was sometimes too low due to the algorithm not always finding all the anomalous data. In the Higgs boson data test case, the estimated proportion of anomalous data was near the real proportion, and it was able to identify the anomalies, but with at least a 20% false positive rate for true positive rates greater than 60%.

One advantage of this approach is that there is a single model parameter, $\lambda$, that directly gives an estimate for the amount of anomalies in the unlabeled data. Besides this, the Gaussian distributions used in the study can be easily replaced with any other parametric distribution. One disadvantage of the algorithm is that it is only able to detect anomalies that manifest themselves as an excess on top of the expected normal data, i.e., it does not discover anomalies where some $\pi_q$ are negative. Another shortcoming is the fact that the higher the dimensionality of the data, the larger the number of observations required to achieve density estimates of certain precision. Thus, it is needed a suitable dimensionality reduction method. As with other mixture models, there is also the hard problem of determining the number of components [62], which in the study was resolved by using the cross-validation-based information criterion.

### 2.3.2 Diagnosing Network-Wide Traffic Anomalies using PCA

As seen in [35], PCA can be used for unsupervised detection of traffic volume anomalies, that is, a sudden positive or negative change in an Origin-Destination (OD) flow’s traffic. This method is based on a separation of the high-dimensional space occupied by a set of network traffic measurements into disjoint subspaces corresponding to non-anomalous and anomalous network conditions, using PCA. For validation, data collected on two different backbone networks was used. The technique not only tries to detect anomalies, but also to identify and quantify them. Here we analyze the detection method, as it is the most relevant for this work.

This method takes as input a $t \times m$ traffic matrix $Y$, being $t$ the number of 10 minute bins measured, and $m$ the number of links in the network. Here, each column $i$ denotes the timeseries of the $i$-th link and each row $j$ represents an instance of all the link’s byte counts at time $j$. After normalizing the matrix, ensuring its columns have zero mean, PCA is applied as described in Section 2.2.2.

Being $u_i$ the projection of the data onto the principal component $v_i$, an empirical threshold is set such that all the $u_i < u_T$ belong to the normal set, with corresponding principal components $V_1$, and all the $u_i < u_T$ belong to the abnormal set, with correspondent principal components $V_2$. In this case the separation procedure consisted in examining the projection on each principal component in order; as soon as a projection was found that contained a $3\sigma$ deviation from the mean, that principal component and all subsequent components were assigned to the anomalous subspace. All previous components were assigned to the non-anomalous subspace. The link traffic vector $y$, that is, a certain column of $Y$,
is then projected to obtain $y_1$ and $y_2$, respectively:

$$y_1 = V_1 V_1^T y$$

and

$$y_2 = (1 - V_1 V_1^T) y .$$

In this way, $y_1$ will contain the modeled traffic and $y_2$ the residual traffic, and thus:

$$y = y_1 + y_2 .$$

In general, the occurrence of a volume anomaly will tend to result in large change to $y_2$. The anomaly is finally detected if the squared prediction error (SPE) $\| y_2 \|^2$ is larger than a threshold determined by the required confidence level using the statistical test called Q-Statistic, given in [63]. This threshold is denoted as $\delta_\alpha^2$, and corresponds to the threshold for the SPE at the $1 - \alpha$ confidence level. This method, when applied to network-wide traffic [35], can also identify the underlying OD flows that are the sources of the anomalies and estimate the amount of traffic involved in the anomalies. For the evaluation of the technique, both data with real and synthetic anomalies was used. The results for the three different data sets with actual anomalies used for validation are shown in Table 2.1.

**Table 2.1:** Results from actual anomalies diagnosed at 99.9% confidence level.

<table>
<thead>
<tr>
<th>Data set</th>
<th>Detection</th>
<th>False Negatives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sprint-1</td>
<td>4/5</td>
<td>6/1003</td>
</tr>
<tr>
<td>Sprint-2</td>
<td>3/4</td>
<td>4/1004</td>
</tr>
<tr>
<td>Abilene</td>
<td>2/3</td>
<td>13/1005</td>
</tr>
</tbody>
</table>

For the two synthetic volume anomalies data sets, Table 2.2 shows the obtained results.

**Table 2.2:** Results from actual anomalies diagnosed at 99.9% confidence level.

<table>
<thead>
<tr>
<th>Data set</th>
<th>Detection</th>
<th>False Negatives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sprint</td>
<td>93%</td>
<td>15%</td>
</tr>
<tr>
<td>Abilene</td>
<td>90%</td>
<td>5%</td>
</tr>
</tbody>
</table>

The previous approach may be extended to traffic features such as source and destination IP addresses and source and destination port numbers, instead of traffic volumes. By substituting the volume matrix $X$ by a traffic feature entropy matrix $H$, minor volume changes such as port scans, small DoS attacks, and other anomalies that have impact on traffic feature distributions can be detected. This couldn’t be achieved with volume-based anomaly detection, which detects anomalies that have large impact on traffic volume. Therefore, the two approaches are complementary.

The computational complexity for this method depends on the matrix $Y$ dimensions, being $O(tm^2)$.
2.4 Feature Selection

Due to the big size of the data set and depending on the CPU or Graphics Processor Unit used, training and cross-validating the models can take up too much time. Feature selection, sometimes called “dimensionality reduction”, can solve this problem, and also possibly improve the results from a learning model.

Dimensionality reduction can be made based on feature transformation, with techniques such as Principal Component Analysis, or compression, using information theory. The resulting representation of the features by these techniques cannot be interpreted by a domain expert. On contrast, feature selection (or pruning) maintains the original interpretability of the features by simply selecting a subset of them. Feature pruning can be used to reduce the training time and to obtain a better generalization behavior of the obtained rules. For neural networks this is specially useful, as their optimization function is non-convex, which allows for it to become stuck in local minimum. Due to this, as less useless features are present in the data set also less local minimum combinations become possible.

Tree-based estimators such as decision trees can estimate feature relevance and discard the irrelevant ones. They classify instances by sorting them through tree-like graphs based on the values of their features [32]. Each internal node in the tree represents a feature, and each branch represents a value that the feature can assume. The instance goes through the corresponding branches until arriving at the final nodes, or leaf nodes, which represent its predicted class. The search for the most relevant features is embedded in an extremely randomized tree classifier, which allows for the modeling of the feature dependencies although it has the disadvantages of being dependent on the classifier and of assuming that instances belonging to different classes will have at least one feature with different values [33].

Decision trees are supervised prediction models that consist in the mapping of the features to the instance’s class or output, splitting the training data set into subsets based on the values of each feature. In this case the quality of these splits is determined by the Gini impurity $I_G$, which can be seen as the measurement of how frequently a randomly chosen feature $i$ from the data set would be wrongly labeled if it were randomly labeled according to the distribution of labels $\{1, 2, ..., m\}$ in the candidate subset $f$. It is given by:

$$I_G(f) = \sum_{i=1}^{m} (1 - f_i)$$  \hspace{1cm} (2.26)

where $f_i$ is the fraction of features belonging to the class $i$. Logically, the desired subsets should have a Gini impurity score as close to zero as possible.

The Extra-Trees algorithm [70] builds an ensemble of unpruned decision trees according to the usual top-down approach. The final prediction is obtained by the majority vote of all the decision trees. It differs from most tree-based ensemble methods by using the full training set to grow the trees (instead
of bootstrap replicas), and by choosing the cut-point values of features for the splitting in an uniformly random way. The first difference is done to minimize the bias, while the second one is done to minimize the variance, comparing to other methods.

The dependency, which is common to all embedded feature selection techniques, in this case means that it will not search for non-linear relations between features. Different feature selection methods exist, like for example wrappers, (e.g. Sequential backward elimination, Genetic algorithms) which are computationally more expensive and have a higher risk of overfitting. Other popular methods are filters (e.g., Fast correlation-based feature selection, i-test), which ignore interaction with the classifier [70].

2.5 Monitoring Wireless Networks

Monitoring wireless networks is a task usually applied by using third-party monitoring applications such as Nagios ², which collect data related to network services and devices to generate alerts. The information used to generate these alerts can be gathered from analyzing the contents of a packet in the network (i.e., deep packet inspection) or from the network devices. For this work, deep packet inspection would not provide much useful information, as it would not be originating from the device, and would depend on always having an user to generate the network packets, so device related data was used instead.

2.5.1 Simple Network Management Protocol

Simple Network Management Protocol (SNMP) is a standard protocol for managing devices on IP networks, used to monitor and configure these devices. It is defined by the Internet Engineering Task Force (IETF) in the Request for Comments (RFC) specifications [6]. SNMP presents management data in the form of variables on the managed devices [64]. A SNMP manager may query an SNMP agent, which is program installed within the monitored device and that collects its information, for a variable. Some of these variables may also be set by the manager. There are several data types for a variable, for example:

- Octet strings, which consist in a string of zero or more bytes, used to represent text strings or physical addresses.

- Gauge32, which are a 32-bit positive number with maximum value of $2^{32} - 1$, used for example for an interface speed on a router.

- Counter64, a 64-bit positive number with maximum value of $2^{64} - 1$, which increases monotonically until reaching its maximum, when it wraps back to zero and repeats the process. It is also set to

²http://www.nagios.org/
zero when the agent is rebooted. Its use is in the deltas, calculated by taking the difference of two values obtained by two successive queries over a certain time interval. It is used for example to measure the traffic on a network interface.

The information is stored and retrieved as defined in the Management Information Base (MIB) files, which define the hierarchical structure and syntax of MIB objects, i.e., the variables, identified by object identifiers (OIDs). MIB files also translate the OIDs, which are numeric, into textual names, for simpler comprehension. SNMP has three different main versions. For this work SNMPv2c, defined in RFCs [65] and [66], will be used, which has the particularity of supporting 64 bit counters, besides 32 bit ones.

Some of the metrics measured in this work are directly related to the performance of a device in a wireless network, such as successful and failed request to send (RTS) frames for an access point. Others, such as the current number of wireless clients associated with the device on a specific interface, might not be directly related but may provide useful information in discovering anomalies by being correlated with other information. On the other hand, some of the SNMP parameters may contaminate the datasets, making it much harder or even impossible for some machine learning techniques to develop a successful classifying model.

Some MIBs are proprietary and provide very specific information about the hardware of a device, while others are public and more widely implemented. In this work all MIBs available for each device that could contain relevant information were analyzed.

2.6 Summary

The problem of identifying network anomalies unrelated to security through machine learning techniques has not been thoroughly studied. Depending on the approach type (supervised, semi-supervised and unsupervised) many options are presented. Classification state-of-the-art results are achieved with deep neural networks and support vector machines, but their training is very slow. On the unsupervised approaches, all require specifying the number of components and are of difficult evaluation. K-Means and k-Medoids both have relatively worse performance in high-dimensions than EM with GMM. PCA has been used with success for network anomaly detection before, and robust alternatives of this method are an interesting option, along with kernel PCA, which often achieves better results than standard PCA in high-dimensions. Semi-supervised machine learning is also possible using both SVMs and EM with GMM. Table 2.3 summarizes the benefits and disadvantages of the various techniques studied in this
Table 2.3: Summary of analyzed techniques.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Type</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM</td>
<td>S/SS</td>
<td>Global minimum; Fast to predict; SOA results</td>
<td>Slow to train</td>
</tr>
<tr>
<td>Deep NN</td>
<td>S</td>
<td>Fast to predict; SOA results</td>
<td>Local minimum; Slow to train</td>
</tr>
<tr>
<td>EM GMM</td>
<td>U/SS</td>
<td>Better clustering in high-dimensions; Fast to test new examples</td>
<td>Local minimum; Needs defining number of components</td>
</tr>
<tr>
<td>PCA</td>
<td>U</td>
<td>Good results in high-dimensions†</td>
<td>Needs defining number of components; Sensitive to outliers*</td>
</tr>
<tr>
<td>K-Means</td>
<td>U</td>
<td>Low running time; Simple</td>
<td>High sensitivity to outliers; Bad performance with many features; Local minimum; Needs defining number of components</td>
</tr>
<tr>
<td>K-Medoids</td>
<td>U</td>
<td>Low sensitivity to outliers; Simple</td>
<td>Local minimum; Bad performance with many features; Needs defining number of components</td>
</tr>
</tbody>
</table>

Legend: S - Supervised; SS - Semi-Supervised; U - Unsupervised; † - Kernel PCA; * - Not true for Robust PCA.
3

Requirements and Solution

Contents

3.1 Approach ................................................. 33
3.2 Requirements ............................................. 34
3.3 Architecture ............................................... 34
3.4 Solution ................................................... 36
3.1 Approach

The first step will consist in collecting data from network devices to create the training, validation and test sets, after proper pre-processing. This data collection can be done through SNMP, as it provides both device-specific and network information. These SNMP values will be retrieved every 15 seconds, from an AP in the Instituto Superior Técnico's wireless network, more specifically, from its Taguspark campus. Thus, this would be the minimum amount of time in which an anomaly will be detected with this system. The SNMP values correspond to 71 different features, such as the counting of failures in receiving Clear to Send (CTS) packet in response to a Request to Send (RTS) packet, which could indicate problems in the Media Access Control (MAC). Other features, such has the overall central processing unit (CPU) busy percentage of the device in a specified period of time, may not be directly related to anomalies in the network but can possibly provide useful information when correlated with more features.

Since the data distribution is certainly correlated with the time of the day and day of the week, another variable is added that translates this information into a value from 0 to 7 (i.e., 0.5 would be Monday at 12:00).

The network conditions in which this data will be gathered will vary on purpose. Data will be gathered on "normal" conditions, that is, when all the currently implemented monitoring systems as well as the users do not detect any anomaly on the network. Data will also be gathered when artificial anomalies are injected, which are expected to approximate the conditions found with real anomalies. These anomalies include: setting the antenna power of the AP to a very low value; covering the AP with a reflective material such as aluminum foil; and creating radio interference by positioning another functioning AP nearby the monitored one.

All the collected data will then require pre-processing, which in this case will consist in its transformation to the various input formats from the different used algorithms, resulting in a matrix where each instance collected is a row and each column a feature. The values in the matrix as a whole will be standardized so that no feature can have much bigger values than any other, since that could result in skewed results for certain algorithms. This standardization consists in, for each feature measured, subtracting its mean and dividing by its standard deviation.

This step also includes feature selection, that is, trying to define the relevant features for this problem.

After this, a ML algorithm is used to create a model, using the training, validation and test sets. This model can later be used to identify the network status, using as input the SNMP values retrieved from the desired AP in the last 15 seconds.
3.2 Requirements

The desired system has the following requirements:

- Should be able to perform SNMP data collection on an AP, given its IP address.
- Should be able to pre-process that data accordingly to the desired type of ML algorithm (supervised or non-supervised).
- Should be able to train and save for future use a desired ML model, with a given input of pre-processed data, and also measure its performance.
- Should be able to identify the network status of a network with a previously trained model, in less than five minutes.
- Should be easy to use, so that it could be used by a network administrator or someone who wants to further research this problem.

3.3 Architecture

Here is presented a system architecture consisting of several modules to fulfill the objectives enumerated in Section 1.1. The modules consist in Data Collection, Data Pre-processing, Model Creation and Performance Measurement, and, finally, the Network State Identification. In the above Figure 3.1 the diagram for this system is presented.

In the Data Collection Module the user can choose if the data to be collected corresponds to a non-anomalous network, an anomalous network, or if the state is unknown. The user can also specify the time duration of the data collection and which device is to be monitored, by specifying its IP address.

In the second module, Data Pre-processing, is where the data has its values normalized and is separated into training, validation and testing sets. The user must specify if the data is to be used with a multi-class or a one-class classification algorithm, since in a multi-class algorithm both anomalous and non-anomalous data are used in all sets, and in a one-class algorithm only in the testing set is the anomalous data used. The user can specify as many data files as he wishes to compose the resulting data set, and can also give it a name.

In the third module, called Model Creation and Performance Measurement, a machine learning algorithm is used to train, save, and test a model using the collected data. The user only has to specify which data set matrix (previously created in the Data Pre-processing module) to use, besides the desired machine learning algorithm. The available algorithms, which are further described in Section 3.4.3, are:
Figure 3.1: Diagram of the implemented system.
• Deep Neural Networks.
• Support Vector Machine.
• One-class Support Vector Machines.

The models are trained with their corresponding libraries using 5-fold cross-validation to determine the best combinations of parameters, which can be hundreds in some cases, and can take from seconds to weeks, depending on the data size. They are then saved for future use, and their performance measured through the display of the precision, recall and F1 scores, as well as the absolute numbers regarding the correct identifications for the anomalous and the non-anomalous instances in the test set. A ROC graph is also displayed, with the value for the AUC. All these performance measures are explained in Section 2.1.1.

In the fourth and final module, Network State Identification, all the other modules are almost completely integrated, but using just a very small dataset, meant to assess the state of the network during the last 45 seconds, that is, using three instances of consecutive SNMP data. For this, data corresponding to that period is first collected, then pre-processed, creating the small data set to be used with a previously saved machine learning model, which then identifies the state of the network based on the majority of each of the decisions for the three data instances. The user can choose which device to monitor by specifying its IP address, choose the type of machine learning algorithm to use and the desired previously created model, and then start receiving continuously in its interface the identified current state of the network.

3.4 Solution

The provided solution featured a command-line interface, implemented in Bash command language, which allowed the use of any of the previously referenced modules. In the next subsections are explained the details of the implementations of these modules.

3.4.1 Data Collection

The SNMP data collection is done through the use of `snmpget` and `snmpwalk` commands on a physical machine connected to the network. The complete list of SNMP parameters collected during the course of this work can be found in Annex A. The queried AP’s IP is specified by user input, as well as the duration of the data collection and the label to be assigned to it. Then the data is stored in the SNMP folder with all this information provided by the user in its file name, plus an added timestamp, to ensure uniqueness.
3.4.2 Data Pre-processing

After being collected, the data must be transformed into a matrix capable of being input to each of the machine learning algorithms. The user is asked to choose the file with the SNMP data to be transformed.

The first step in this transformation is the parsing of the SNMP values, which was done in the Python programming language. This first step had some particularities, such as sometimes SNMP values being skipped, possibly due to communication errors between the monitoring and monitored devices. Other problem which sometimes occurred was a client dropping or connecting after the client-related SNMP parameters started being collected but had yet not finished. Lastly there was a specific behaviour to take into account with Counter SNMP parameters (explained in Section 2.5.1): for consecutive queries on these objects if the last value obtained, \( A \), is less than the one before it, \( B \), then the counter meanwhile reached its maximum and wrapped to zero. For this reason if the counter is 32 bits one must add \( 2^{32} \) to \( A \) before calculating the corresponding delta, and \( 2^{64} \) in the case of the 64 bits counter. The resultant file from this step is a matrix with real values, one line per each set of SNMP parameters collected every 15 seconds and one column per type of SNMP parameter.

These values have yet to be normalized and separated in training, (possibly) validation and test sets before being fed to a machine learning algorithm, which is done in the following step, implemented with the GNU Octave programming language. First, the user chooses if the data set is to be used with a multi-class or one-class classifier. In the first option the data is separated in all three sets, while with the second option only the training and test sets are created. All the lines in each matrix are randomly reordered between all data sets, to achieve better generalization results, and it is ensured that test set represents 20% of all the data. The data is normalized to be zero-mean by subtracting the mean, and to have unit variance by dividing it by its standard deviation. The validation and test sets use the mean and standard deviation from the training set, which overcomes problems such as overfitting (explained in Section 2.2.1). The output from this step is a Matlab equivalent GNU Octave matrix file. The user then gives a name to the created file.

3.4.3 Model Creation and Performance Measurement

The model creation depends on the learning algorithm. As seen before, each machine learning approach has its advantages and disadvantages. Since the collected data in this work will be both labeled (as either “normal data” or an “anomaly data”, the latter consisting in the group of the injected artificial anomalies), and since they are typically the best performing algorithms, given sufficient data, supervised learning models will be generated. In the approach, both the multi-class and the one-class classification variants will be explored.

For the multi-class classification option, current algorithms that achieve the best performance are
regularized deep neural networks with rectifier or maxout activation functions, and SVMs. Each of them has specific advantages and disadvantages, and it is hard to guess which will perform better for a new problem, so both are available. For the one-class classification alternative, a technique requiring a "normal" labeled data set will be used: one-class support vector machines.

For the training of a Neural Network model, several implementations were considered, and two tested. Fast Artificial Neural Network (FANN)\(^1\) is an open source, free library which implements multi-layer networks in the C programming language, with bindings for more than 20 programming languages available. It features backpropagation training, several activation functions, is relatively fast to train and execute, easy to use and has graphical or command-line interfaces available. However, for this work, the main requirements of a library are a very fast identification time, the ability to integrate in a command-line interface and the best possible results. This implementation didn’t achieve the latter one, possibly due to its lack of more state-of-the art features such as dropout and rectified linear units, analyzed in Section 2.2.1. After a week of cross-validation and following testing, it became obvious that the results weren’t satisfactory, as it achieved around 70% for each artificial anomaly detection. The tool was used with Rprop (short for "resilient backpropagation") as training method, Gaussian hidden activation functions, and Linear Piece Symmetric as out activation functions, being all these parameters chosen by the cross-validation. Possibly a larger data set would achieve better results, but that would be time consuming and an alternative and better implementation was found.

Nolearn\(^2\) is a python wrapper for deep neural network libraries, specifically the Deep Convolutional Activation Feature for Generic Visual Recognition (known also as DeCAF) [71] and gdbn\(^3\).

DeCAF was built specifically for the image recognition domain, and using it would necessarily mean to pre-process the data the same way images are expected to be pre-processed, losing relevant information. Besides that, taking into account that this neural network uses several characteristics only found in images, it wouldn’t make sense to use it. The gdbn, on the other hand, features dropout, rectified linear units in the hidden layers, various activation functions (including the more common ones such as sigmoid or linear) for the output layer and the possibility of pre-training. This implementation, besides having a very fast execution time and possibility of command-line interface integration, also achieved excellent results for the artificial anomalies. It also features many state-of-the-art methods for classification in neural networks.

For the training of the SVM model, mainly three different implementations were considered, based on their varying characteristics: SVM-Light [72], LIBSVM [17] and LIBLINEAR [73].

LIBLINEAR is an open source library that has the theoretical limitation of only solving linear problems, but is capable of training a much larger data set (feature and instance wise) due to this, and generally in

\(^1\)http://leenissen.dk/
\(^2\)https://pythonhosted.org/nolearn/
\(^3\)https://github.com/dnouri/gdbn
a much faster way. As the large dataset was a problem in other implementations (that even led to feature selection), this library was tested. A python wrapper was used, called scikit-learn, which also eased the k-fold cross-validation process, performance measurements and model persistence.

LIBSVM is an open source library that can solve non-linear problems through the use of kernels, as referenced in Section 2.2.1. It supports probability estimates, various kernels and multi-class classification, and naturally has a different optimization objective than the LIBLINEAR library. Once again the python wrapper scikit-learn was used, along with some functions external to the LIBSVM library.

SVMLIGHT is another open source library that follows different optimization algorithms than the other methods \(^4\). It is slower than LIBLINEAR in the training phase, but faster than LIBSVM while still supporting non-linear kernel functions. For this work a python binding\(^5\) was used, which facilitated the integration with the performance analysis.

Between these three SVM implementations, the first best results were obtained with LIBSVM, although only slight. Adding to that, the sickit integration with this library also allowed for easy model saving and performance analysis, so it was chosen as the implementation for the multi-class and one-class classification SVM models.

In this module, the user selects a file from the ones created in the Data Pre-processing module, and one of the learning algorithms from the implemented, as referenced above. Using Python, each training algorithm was programmed to try a large range of values for its specific hyperparameters using cross-validation. After the best performing set of hyperparameters and respective model are found, its ROC graph and scores are generated for analysis by the user. The scores include the AUC, the precision, recall and F1 score. The model is saved and available for future use.

### 3.4.4 Network State Identification

Here, parts of all the previous modules are used to assess the state of the network in near real-time. The user selects the IP of the network device it wants to monitor, and one of the previously saved models to make this determination. Three consecutive SNMP data collections, spaced by 15 seconds, are made, and then normalized using the mean and standard deviation of the selected model’s training set. The selected model then determines the state of each SNMP data entry, and the majority is output via the command-line interface and to a text file. The process is repeated indefinitely.

\(^4\) [http://www.cs.cornell.edu/People/tj/svm\%5Flight](http://www.cs.cornell.edu/People/tj/svm%5Flight)

\(^5\) [https://pypi.python.org/pypi/svmlight/0.4](https://pypi.python.org/pypi/svmlight/0.4)
4

Data Collection

Contents

4.1 Obtaining the ground-truth ........................................... 43
4.2 Collected Data .......................................................... 45
4.3 Feature Selection ....................................................... 46
4.4 Creating the Artificial Anomalies ................................. 48
4.1 Obtaining the ground-truth

A ground-truth in this context is a data set which we know to a great degree of certainty to contain SNMP values pertaining to a network without anomalies. It is necessary to achieve meaningful results with this work, as well as to use in the training of the machine learning models. But obtaining a perfect ground-truth in this scenario is arguably a near-impossible task: it is expensive to gather enough trusted users for such a long time; there can always be undetected attacks to the network; users would probably have to be restricted in their use of applications and websites to visit, in order to avoid possible attacks, which would in turn create an unrealistic scenario; and the monitored Access Point could malfunction without it being necessarily noticed. Nevertheless, and having in mind both these challenges and the limited resources available, two different attempts were made to create a ground-truth data set.

The first attempt was less rigorous, as it simply consisted in gathering the SNMP data in the Tagus-park network when no users reported any noticeable degradation in their quality of experience. This attempt has the advantage of being as real as possible, while having a greater possibility of being contaminated with instances of undetected anomalies.

The second attempt consisted in connecting four laptops and simulating an internet user behaviour in each of them, using only one Access Point to access the network. The operating systems of the devices were Ubuntu Precise Pangolin (12.04). The model of the AP used for the controlled data collection was a Cisco Aironet 1131 AG \(^1\), an enterprise-class access point similar to those used in the other parts of the data collection.

The AP was placed at two meters from the four laptops, in an isolated environment to avoid possible radio interference or occasional obstacles. The collection corresponded to seventeen and a half hours in total, which were obtained in four different days. A viable alternative would be to use the VirtualBox software \(^2\), which permits the creation of several Operating Systems (OSs) in the same physical machine, each having its own virtual environment. With this, one could then simulate the users and their use of the network. It could have the advantage of generating more connected users, but on the other side the Access Point would then not be exposed to several network cards, which would be less realistic. A third approach, complementary to these, could consist in simulating the network itself, increasing even more the control over the collected data and its reproducibility, but this would also bring the relevant disadvantage of creating such uniform and unrealistic data that could not possibly be used by the machine learning algorithms to infer the desired realistic non-anomalous behavior.

Similar applications to those used by real users were used here, in real time, and in a controlled environment, while still collecting the SNMP data from the monitored Access Point. The definition of which applications and services to use didn’t have to be strictly rigorous, as the SNMP data directly


\(^2\) [https://www.virtualbox.org](https://www.virtualbox.org)
influenced by these is limited to the User Datagram Protocol (UDP) protocol. Nonetheless, in order to achieve the correct proportion of UDP packets, and because it is possible that these applications have indirect influence in other SNMP parameters, certain measures were taken to try and imitate real-world behavior regarding network use.

The applications were executed at random times (as opposed to being executed periodically using a job scheduler such as cron), and the frequency for each execution differed accordingly to the estimated traffic proportions to be generated in Table 4.1. In this way, traffic uniformity was avoided, and thus a more realistic simulation was made.

To determine which traffic should be generated the statistics available through [67] were analyzed, more specifically the ones found at the study’s website\(^3\) pertaining to traffic generated in 2014, as these come from an European network that connects around one hundred universities and research institutions, the Danish e-Infrastructure Cooperation. In general, these statistics showed a traffic distribution per IP-level protocol of about 70% of the data for Transmission Control Protocol (TCP), 25% for UDP, and the remaining for other protocols related to VPN use, such as Encapsulating Security Payload (ESP) and Generic Routing Encapsulation (GRE), as seen in Figure 4.1.

The work made in [68] was more closely followed, as it tried to emulate the same network as the monitored one in this work. It concluded that nowadays most traffic consists in web and Peer-to-Peer (P2P) related traffic, as well as a significant emergent portion of video streaming, and a smaller on File Transfer Protocol (FTP). Based on this, Table 4.1 presents the protocols and corresponding traffic generated in bytes, used by the virtual hosts:

---

\(^3\)http://stats.simpleweb.org/statistics.php?l=10
Table 4.1: List of applications and their traffic proportions.

<table>
<thead>
<tr>
<th>Protocols (Applications)</th>
<th>Percentage of total bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP/HTTPS (web browsing)</td>
<td>40%-50%</td>
</tr>
<tr>
<td>HTTP (streaming)</td>
<td>15%-25%</td>
</tr>
<tr>
<td>BitTorrent (file transferring/sharing)</td>
<td>20%-30%</td>
</tr>
<tr>
<td>FTP (file transferring)</td>
<td>5%-15%</td>
</tr>
</tbody>
</table>

4.2 Collected Data

The data collection firstly consisted of the 59 SNMP parameters listed in Annex A.1. As each interface of the AP has some of those SNMP parameters unique to it, such as the total number of octets received on it, the total parameters count is extended 71. The restriction to only SNMP parameters is done in order to provide a solution to this problem that can be deployed without additional network infrastructure, using parameters widely available in most Access Points in the industry. Figure 4.2 represents the basic setup required for the data collection.

![Figure 4.2: Diagram of setup for the data collection.](image)

Briefly explaining the parameters, those consisted in almost all the possibly relevant measurements, indicators or counters provided by the Access Point, such as:

- Traffic related counters, which could be the number of UDP datagrams or all the octets in or out a specified interface.

- Errors, such as Frame Check Sequence Errors, failures in delivering UDP datagrams, and Temporal Key Integrity Protocol (TKIP) errors, or failures in receiving Clear to Send frames in response to Request to Send frames.

- Percentages, such as the average CPU busy percentage in a specified interval.
• The reasons of the last monitored 802.11 Disassociated, Deauthentication or failed Authentication frames.

• Various other counters and measurements, like for example the number of unused memory bytes in the managed device, the write retries in the chip, or the stations associated with the AP.

All these parameters were monitored for the two active interfaces of each AP, separately.

Besides those, other 12 parameters, specified in Annex A.2, were collected after the feature selection explained in Section 4.3. These were directly related to the associated wireless clients and derived by the information provided by them. They were:

• Signal quality and signal strength, derived from the last packet received by the client.

• The number of packets and bytes sent and received, as well as duplicate packets.

• The number of successful and failed Media Access Control Service Data Units transmissions.

• The number packets which failed to be properly decrypted via the Wired Equivalent Privacy (WEP) security mechanism.

• The number of Message Integrity Code errors and missing frames for each client.

The above values were always averaged by the number of connected clients at the time of the data collection, as they would not make sense without that correlation for the purpose of generalization through the machine learning models.

### 4.3 Feature Selection

For this project it became needed to reduce the data set dimensionality, as training the models was taking too much time and the performance could improve with it.

In this work, feature selection was made by using the Extra-trees algorithm, explained in Section 2.4. The algorithm used is part of the Scikit-learn python package[^4].

The feature scoring obtained consists in a ranking of all of the 71 SNMP variables monitored that were not specific to each client, being the top 12 shown in the Table 4.2.

Some of the features among the the top scoring ones were somewhat surprising: the fourth ranking feature indicates the number of bytes from the memory pool that are currently unused on the managed device. The feature at second place indicates the total number of received UDP datagrams for which there was no application at the destination port, indicating the ability of the system, which only receives SNMP data at the device level, to detect anomalies affecting the network at the application level.

Table 4.2: The 12 top scoring features using the extra-trees algorithm.

<table>
<thead>
<tr>
<th>Ranking</th>
<th>OID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1º</td>
<td>1.3.6.1.4.1.9.9.273.1.1.2.1.1</td>
<td>Number of wireless clients currently associating with the device on the monitored interface</td>
</tr>
<tr>
<td>2º</td>
<td>1.3.6.1.2.1.7.2</td>
<td>Number of received UDP datagrams for which there was no application at the destination port</td>
</tr>
<tr>
<td>3º</td>
<td>- (Custom)</td>
<td>Value indicating the day of the week and hour</td>
</tr>
<tr>
<td>4º</td>
<td>1.3.6.1.4.1.9.9.48.1.1.1.6</td>
<td>Number of bytes from the memory pool that are currently unused on the managed device</td>
</tr>
<tr>
<td>5º</td>
<td>1.3.6.1.4.1.9.9.109.1.1.1.1.10</td>
<td>Overall CPU busy percentage in the last 15 seconds</td>
</tr>
<tr>
<td>6º</td>
<td>1.3.6.1.4.1.9.9.48.1.1.1.5</td>
<td>Number of bytes from the memory pool that are currently in use by applications on the managed device</td>
</tr>
<tr>
<td>7º</td>
<td>1.2.840.10036.2.2.1.12</td>
<td>Counter of FCS errors in a received MAC Protocol Data Unit (MPDU)</td>
</tr>
<tr>
<td>8º</td>
<td>1.3.6.1.4.1.9.9.272.1.2.1.1.1.2</td>
<td>Counter of MAC Cyclic Redundancy Check (CRC) errors</td>
</tr>
<tr>
<td>9º</td>
<td>1.3.6.1.4.1.9.9.48.1.1.1.7</td>
<td>Number of contiguous bytes from the memory pool that are currently unused on the monitored device</td>
</tr>
<tr>
<td>10º</td>
<td>1.3.6.1.2.1.7.4</td>
<td>Number of UDP datagrams sent from the monitored device</td>
</tr>
<tr>
<td>11º</td>
<td>1.3.6.1.2.1.7.1</td>
<td>Number of UDP datagrams delivered to UDP users</td>
</tr>
<tr>
<td>12º</td>
<td>1.3.6.1.2.1.2.2.1.1.11</td>
<td>Number of subnetwork-unicast packets delivered to a higher-layer protocol.</td>
</tr>
</tbody>
</table>

The subset of the first 12 SNMP parameters was included in the new version of the data set to train and test the models. This number was based on the knee of the graph containing the feature scores, as shown in Figure 4.3.

It is important to notice, though, that the optimal subset of features relevant for the artificial anomalies is not necessarily the same optimal subset to detect real anomalies. The method used for the above feature ranking also does not take into account non-linear dependencies between features. Besides that, anomalies not present in the training set could affect other features. For those reasons some features were also included in the final data set definition, either because they could indicate a radio or a device problem. Table 4.3 presents some of them and their corresponding added value.

Finally, all the client specific parameters briefly described in 4.2 and specified in Annex A.2 were also added to the final data set, as they provide direct information about the users who may experience a degraded quality of experience. This resulted in a final number of 42 different parameters in the final data set.
Table 4.3: Added features for the data set.

<table>
<thead>
<tr>
<th>Type of problem it could indicate</th>
<th>OID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radio or device</td>
<td>1.2.840.10036.2.2.1.8</td>
<td>The number of times when a CTS was not received in response to a RTS.</td>
</tr>
<tr>
<td>Radio or device</td>
<td>1.3.6.1.2.1.2.2.1.19</td>
<td>The number of outbound packets which were chosen to be discarded even though no errors had been detected to prevent their being transmitted (possibly to free buffer space).</td>
</tr>
<tr>
<td>Related to an external attack on the network</td>
<td>1.3.6.1.4.1.9.9.273.1.1.4.1.3</td>
<td>The number of Message Integrity Check (MIC) failures reported by clients on this radio interface since device re-started</td>
</tr>
<tr>
<td>Related to an external attack on the network</td>
<td>1.3.6.1.4.1.9.9.273.1.1.4.1.6</td>
<td>The number of TKIP replay errors detected on this interface since device re-started</td>
</tr>
<tr>
<td>Transmission</td>
<td>1.3.6.1.2.1.7.3</td>
<td>The number of received UDP datagrams that could not be delivered for reasons other than the lack of an application at the destination port</td>
</tr>
<tr>
<td>Transmission</td>
<td>1.3.6.1.2.1.2.2.1.20</td>
<td>Number of outbound packets that could not be transmitted because of errors</td>
</tr>
</tbody>
</table>

4.4 Creating the Artificial Anomalies

Three different artificial anomalous situations were created during the data collection for the training set. These anomalies are expected to approximate the conditions found with real device-related anomalies in a network. The anomalies were put into action in an university library’s network (of the Taguspark Campus of Instituto Superior Técnico) used by dozens of students everyday, who usually connect to it by laptop or smartphones.

The first of these anomalies consisted in setting the antenna power of the Access Point to a very
low value. The rationale behind it is that when the power signal degrades for some reason, being it due to faulty equipment or inadequate power supply, for example, it is usually a gradual change that is not rapidly detected by common network monitoring tools or even by the users. The Access Point was configured to have an antenna power of 7dbm for one week, and users reported a deterioration of the network’s quality of experience. Analyzing the collected data and comparing it to the data corresponding to the unchanged network it was seen that almost all the parameters differed by greater than 10% of their original values. The only parameters that remained almost unchanged were related to the memory use of the device (besides the parameter indicating the day of the week and the time of the day).

Another artificial anomaly created consisted in covering the Access Point with a reflective material, in this case aluminum foil. Often in real-case scenarios APs are inadequately positioned near reflective material, or others obstacles are put near an AP after its installation, and the situation may not be obviously detected by the network administrator, requiring an user complaint. In this case, the users also noticed a degradation of the network’s quality of experience. Most parameters had only a slight change relatively to the data without anomalies (i.e., less than a 5% difference). The number of wireless clients associated with the device was where differences were most significant, being ten times less when the anomaly was injected. Interestingly, the overall CPU busy percentage was the second most deviated parameter, at approximately 5.5 times less, followed by the SNMP value that indicates the Reason Code in a 802.11 Disassociation frame, and the value that holds the most recently transmitted Status Code in a failed 802.11 Authentication frame.

Interference in real-case scenarios can be caused by other electronic equipment using the same frequency, specially at the 2.4Ghz, but already many other devices exist that operate at 5Ghz, such as cordless phones, perimeter sensors or radar. With the each day more common presence of wifi-capable devices, some interference problems may also happen due to bad implementations of the Institute of Electrical and Electronics Engineers (IEEE) 802.11 specifications for media access control and the physical layer. The last artificially injected anomaly was created by positioning another functioning AP nearby the monitored one, on an overlapping channel of the monitored one. Constant traffic was generated between the newly put AP and a connected user using the iperf tool, creating interference with the monitored AP. The parameters that most deviated relatively to the data corresponding to a network in a normal state were once again the number of wireless clients associated with the device, being 20 times less; the overall CPU busy percentage, at approximately 3.9 times less; the Reason Codes of the 802.11 Deauthentication and Disassociation frames, and the Status Code of the 802.11 Authentication frame.
Experimental Assessment

Contents

5.1 Time to train and execute ............................................. 53
5.2 Big data set results ................................................. 54
5.3 Small data set results with no model retraining ................. 57
5.4 Small data set results using retrained models .................. 63
In this section there will be presented and analyzed the results obtained that are independent from the data, and also those obtained using two different data sets (always with the same machine). The two data sets were created in similar conditions, being their only differences a distance of two months and the fact that the second one amounted to a few days less of collected data. For this reason the first one will be called “big data set” and the second “small data set”. The big data set consisted SNMP collected data at every 15 seconds for a total duration of two weeks, and the small to nearly 12 days. The SNMP data consisted of 42 parameters, achieved as explained in Section 4.3, and half of the data corresponded to the network in the normal state, while the other half corresponded to the network in an artificially anomalous state, composed by equal parts of each of the three anomalies. This amounted to a total of roughly 80000 observations for the first data set and 65000 for the second. The monitoring physical machine used for the tests had an Intel Core i7-3630QM CPU, 8 GB RAM, and was running Ubuntu 12.04. The monitored device was an Access Point in the Taguspark library.

5.1 Time to train and execute

The timing objective in this work was to successfully diagnose the state of the network in the last 15 seconds in a timely manner, that is, in a few minutes at most. But the registered training time took a day, at minimum, for each model. The objective was then achieved by firstly training the models and then saving them for future use, at the cost of relatively small disk space.

<table>
<thead>
<tr>
<th>ML Model</th>
<th>Time to identify in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervised Support Vector Machines</td>
<td>93</td>
</tr>
<tr>
<td>Neural Networks</td>
<td>98</td>
</tr>
<tr>
<td>One-class SVM</td>
<td>90</td>
</tr>
</tbody>
</table>

*Table 5.1: Average time that each ML model took to identify the network status.*

After this, the only time-consuming task is to run the desired model with the collected data. Table 5.1 shows the average time it took for each of the machine learning models used by the system to identify the network status. Each identification was based on the majority obtained from three consecutive SNMP data collections, spaced by 15 seconds. Besides being collected, the data had to be converted to a normalized matrix (using the mean and sigma derived from the training data set) before being fed to the machine learning model.

<table>
<thead>
<tr>
<th>ML Model</th>
<th>Time to train in days</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervised Support Vector Machines</td>
<td>7</td>
</tr>
<tr>
<td>Neural Networks</td>
<td>1.5</td>
</tr>
<tr>
<td>One-class SVM</td>
<td>3.5</td>
</tr>
</tbody>
</table>

*Table 5.2: Average training time for each ML model.*

The training time of the models may be also considered relevant, since the characteristics of what
constitutes a non-anomalous network may be altered as both internet use and Access Points change, bringing a new data paradigm to the field. As can be seen in Table 5.2 Support Vector Machines took the longest to cross-validate, while Neural Networks were the fastest option.

5.2 Big data set results

In this subsection are presented the results achieved with each model for the first and bigger data set, in the form, when possible, of the ROC graph, AUC score, F1 score, precision and recall.

![Figure 5.1: ROC graph for the aluminum anomaly with the NN model.](image)

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-anomalous</td>
<td>0.98</td>
<td>0.84</td>
<td>0.91</td>
</tr>
<tr>
<td>Anomalous</td>
<td>0.86</td>
<td>0.98</td>
<td>0.91</td>
</tr>
<tr>
<td>Total</td>
<td>0.92</td>
<td>0.91</td>
<td>0.91</td>
</tr>
</tbody>
</table>

For the Neural Network models testing for the recognition of normal behaviour or the aggregation of all three anomalies did not provide perfect results, as can be seen in Figure 5.2 and Table 5.4, but were nonetheless very good. In order to understand if there was a particular anomaly more difficult to identify, all three were separated and their identification results compared. Both the anomaly with a lowered signal power and the interference anomaly were completely identified, and so their performance scores were perfect. The aluminum foil anomaly had slightly worse scores, as seen in Figure 5.1 and Table 5.3, and so it can be concluded that it was the major reason for the imperfect scores in the anomaly aggregation tests.
Figure 5.2: ROC graph for all anomalies with the NN model.

Figure 5.3: ROC graph for the aluminum anomaly with the SVM model.
Table 5.4: Scores for the aggregation of anomalies with the NN model.

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-anomalous</td>
<td>0.99</td>
<td>0.90</td>
<td>0.94</td>
</tr>
<tr>
<td>Anomalous</td>
<td>0.90</td>
<td>0.99</td>
<td>0.95</td>
</tr>
<tr>
<td>Total</td>
<td>0.95</td>
<td>0.94</td>
<td>0.94</td>
</tr>
</tbody>
</table>

Figure 5.4: ROC graph for all anomalies with the SVM model.

For the Support Vector Machine models all the performance scores, except for the one with the aluminum anomaly represented in Figure 5.3 and Table 5.5, were also perfect. The performance with all anomalies aggregated was slightly better than that of the NN model, and once again the few identification errors were due to the aluminum-related data. These are shown in Figure 5.4 and Table 5.6.

Table 5.5: Scores for the aluminum anomaly with the Supervised SVM model.

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-anomalous</td>
<td>0.98</td>
<td>0.92</td>
<td>0.94</td>
</tr>
<tr>
<td>Anomalous</td>
<td>0.92</td>
<td>0.98</td>
<td>0.95</td>
</tr>
<tr>
<td>Total</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
</tbody>
</table>

Table 5.6: Scores for the all the anomalies with the Supervised SVM model.

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anomalous</td>
<td>0.98</td>
<td>0.98</td>
<td>0.98</td>
</tr>
<tr>
<td>Non-anomalous instance</td>
<td>0.98</td>
<td>0.98</td>
<td>0.98</td>
</tr>
<tr>
<td>Total</td>
<td>0.98</td>
<td>0.98</td>
<td>0.98</td>
</tr>
</tbody>
</table>

The One-class SVM revealed itself a much poorer method for discovering anomalies. Several combinations of its parameters were compared in order to find the best possible one. These parameters included:
• The type of kernel used: Linear, sigmoid, RBF and poly.

• Nu, an upper bound on the percentage of margin errors and a lower bound on the percentage of support vectors relative to the total number of training examples.

• The tolerance for stopping criterion, which is a limit for the norm of the objective’s gradient.

• The gamma parameter, a coefficient for the RBF, poly and sigmoid kernels.

Despite the extensive grid-search, the best overall F1-score achieved was of only 0.63. Table 5.7 shows the complete results for 20000 test examples. Given the relatively bad results, this algorithm was abandoned in favor of the supervised alternatives. The ROC graph was not provided since the used implementation did not feature a probabilistic output.

| Table 5.7: Scores for the all the anomalies with the One-class SVM model. |
|-------------------|--------|-------|--------|
| Instance Type     | Precision | Recall | F1-score |
| Anomalous         | 0.72    | 0.85  | 0.78   |
| Non-anomalous instance | 0.37    | 0.22  | 0.28   |
| Total             | 0.62    | 0.66  | 0.63   |

The existence of some cases in the aluminum anomaly data set which were not correctly identified by any model indicates that these cases could be in fact very hard or impossible to differentiate, perhaps due to some error in its collection or simply due to indistinguishable cases using only the provided data.

5.3 Small data set results with no model retraining

Two months later a new and smaller data set was created which replicated all the four scenarios (three anomalies and one without any anomaly), and its results using the previously trained models were compared. In spite of the excellent results yielded in the training scenario, which prove there was some discriminating data that differentiated each network scenario, the models could not replicate them just two months later. The gap in time, which was due in part to the closing of the monitored place (a library) for a month, seemed to have created somehow a new paradigm. After extensive tests it was concluded that all models always identified an anomaly in the network, even when there wasn’t one, which made them useless.

This could have happened due to various reasons:

• Extra parameters that did not provide any useful information were in fact lowering the performance of the trained models.

• A different number of users was using the network, which was similar to the one obtained in previous anomalous scenarios.
Different user behavior, as new users came in and the time of the year was also not the same as before.

New network configurations which affected somehow the collected SNMP parameters present in each AP.

Automatic changes in the AP that affected the collected SNMP parameters.

A human error collecting either the previous or the new data sets.

Inquiries were made to determine that no new network configurations were made, and the data from both data sets analyzed and no errors found in its collection or pre-processing. The number of users also did not seem different than before, so it became necessary to find if there were extra parameters present in each collection, if there was new user behavior, or if there were uncontrolled changes in the AP which affected its SNMP parameters.

There were still three possible underlying causes remaining, and in order to explore their importance, the Extra-Trees algorithm referenced in Section 4.3 was once again applied. This time one class was the original data set and the other class the small data set, and in neither of them was the state of the network differentiated. The goal was to find parameters which distinguished the data sets between
themselves and that might be the source of the misclassification. The obtained results indicated in fact some discriminating SNMP parameters, as shown in Figure 5.5.

<table>
<thead>
<tr>
<th>Index</th>
<th>OID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1.3.6.1.4.1.9.9.48.1.1.1.6</td>
<td>Indicates the number of bytes from the memory pool that are currently unused on the managed device.</td>
</tr>
<tr>
<td>4</td>
<td>1.3.6.1.4.1.9.9.109.1.1.1.1.10</td>
<td>The overall CPU busy percentage in the last 15 seconds</td>
</tr>
<tr>
<td>6</td>
<td>1.3.6.1.4.1.9.9.48.1.1.1.5</td>
<td>Indicates the number of bytes from the memory pool that are currently in use by applications on the managed device.</td>
</tr>
<tr>
<td>12</td>
<td>1.3.6.1.4.1.9.9.272.1.2.1.1.1.2</td>
<td>The number of frames received with any MAC CRC error</td>
</tr>
<tr>
<td>13</td>
<td>1.3.6.1.4.1.9.9.48.1.1.1.7</td>
<td>Indicates the largest number of contiguous bytes from the memory pool that are currently unused on the managed device</td>
</tr>
<tr>
<td>42</td>
<td>(Custom)</td>
<td>Value indicating the day of the week and hour</td>
</tr>
</tbody>
</table>

**Table 5.8**: The 6 top scoring features using the extra-trees algorithm.

Table 5.8 indicates for the six most relevant features each OID and its respective description.

As can be seen, these imply not new user behavior but changes in the AP which affected the collected parameters, and that could be affecting its decision-making abilities. Since their removal implied a different input fed to each model, the models were once again trained, this time with only the remaining 35 SNMP parameters. However the new results obtained were really poor, as shown in Figure 5.6 for the SVM model and Figure 5.7 for the NN model.

If extra parameters were present, a way to minimize their importance would be to apply the Principal
Figure 5.7: ROC graph achieved with the NN model and the selected 35 parameters.

Figure 5.8: Scree plot of the percent variability held by each principal component.
Component Analysis technique, as referenced earlier in Section 2.1.2, and so it was applied to the original data set. This reduced the dimension of the input fed to the models, and thus required new models to be trained and generated. Figure 5.8 shows the obtained variance for each accumulated principal component, and as can be seen the first 7 components hold more than 99.5% of the total variance. This was very interesting as it indicated that the total dimension of the data set could be reduced by a factor greater than 5 while having little variance lost. However, one should note that the parameter or set of parameters responsible for a certain decision in machine learning models are not necessarily included in the first principal components.

In order to try to understand the importance of each principal component, various models for gradually more principal components were trained and their performance analyzed. The training data was approximately one fourth of the complete big data set, to reduce the training time, which in theory should reduce the performance slightly. Each model was distinguished by the number of principal components received, which ranged from 1 to 35, at intervals of one for the first 10 principal components, and of five for the remaining. The algorithm used was the SVM, since it previously was the best performing algorithm. Figure 5.9 shows the change in performance with the number of PCs, and interestingly there is a degradation of performance between 20 and 30 PCs, indicating that some possibly confounding variance was added with the variance corresponding to those PCs.

After this, all the models were given the data from the small data set with the PCA technique applied.
Table 5.9: Scores for the SVM model, with data transformed using 13 principal components.

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anomalous</td>
<td>0.69</td>
<td>0.69</td>
<td>0.69</td>
</tr>
<tr>
<td>Non-Anomalous</td>
<td>0.69</td>
<td>0.69</td>
<td>0.69</td>
</tr>
<tr>
<td>Total</td>
<td>0.69</td>
<td>0.69</td>
<td>0.69</td>
</tr>
</tbody>
</table>

Figure 5.10: AUC achieved by each SVM model, by number of PCs used in the data set.

This comparison confirmed that the variance found in the first components is in fact sufficient to create a working classification model even after some unpredictable change in the paradigm, although with worse results than those achieved without the PCA dimensionality reduction. It also indicated that extra parameters were presented, although the reason for the initial performance degradation could be composed by this and the other two remaining causes: new user behavior and uncontrolled changes in the AP.

Having exhausted the most important techniques to identify and remove the possible data affecting the decision-making abilities, the best course of action to improve even more the results was simply to retrain the models with the new smaller data set, and confirm its usefulness in the new paradigm.
5.4 Small data set results using retrained models

After proper retraining with the small data set, all models were once again able to recognize the artificial anomalies, as expected, proving that the poor results obtained earlier depended on the data.

![ROC graph for the second data set, with the SVM model.](image)

**Figure 5.11:** ROC graph for the second data set, with the SVM model.

**Table 5.10:** Scores for the SVM model with the second data set.

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-anomalous</td>
<td>1.00</td>
<td>0.98</td>
<td>0.99</td>
</tr>
<tr>
<td>Anomalous</td>
<td>0.98</td>
<td>1.00</td>
<td>0.99</td>
</tr>
<tr>
<td>Total</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
</tr>
</tbody>
</table>

In order to this time better understand how much data should be collected before a viable model could be achieved, different models were generated with gradually more data. The required data in total hours collected before achieving an F-1 score greater than 75% was of 2 hours for the SVM model and of 27 hours for the NN model. Once again the best results came from the Supervised Support Vector Machine model, as it would be the fastest to adapt when the network conditions are significantly altered, which gives a very significant advantage to this algorithm.

After this the models were trained with the entire data set, consisting of 30 hours of SNMP data. In a first try, the NN model achieved significantly worse results even with all the 30 hours collected: and F-1 score of 0.78 and an AUC of 0.77, while the SVM achieved almost perfect results, as seen in Table 5.10 and Figure 5.11.

In order to try to improve the results of the model produced by this algorithm, the number of support vectors from the new SVM model were counted and given as the number of hidden layers of the NN
Figure 5.12: ROC graph for the second data set, with the NN model.

Table 5.11: Scores for the NN model with the second data set.

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-anomalous</td>
<td>0.97</td>
<td>1.00</td>
<td>0.98</td>
</tr>
<tr>
<td>Anomalous</td>
<td>1.00</td>
<td>0.97</td>
<td>0.98</td>
</tr>
<tr>
<td>Total</td>
<td>0.98</td>
<td>0.98</td>
<td>0.98</td>
</tr>
</tbody>
</table>

training algorithm. This significantly improved the results for the NN model, as shown in Figure 5.12 and Table 5.11.
Conclusions

Contents

6.1 Achievements ................................................................. 67
6.2 System Limitations and Future Work ................................. 67
6.1 Achievements

While today wireless networks are becoming more and more present, it still often takes a lot of time to detect an anomaly that degrades user experience. The objective of this work was to develop a system which could detect these types of anomalies in a timely manner, requiring no additional infrastructure and using Machine Learning methods.

After an extensive study of the main machine learning techniques (and their performance metrics), with a focus on those used to solve similar problems, it was decided that three different ML algorithms would be compared. The algorithms were of two types: the supervised multi-class Support Vector Machines and Neural Networks, and the supervised one-class Support Vector Machines. The collected data would consist of 71 SNMP parameters from the Taguspark network, every 15 seconds. A system was designed with usability in mind, with several modules that performed: SNMP data collection from Access Points; data pre-processing, using octave scripts; model training, saving and performance analysis, using the aforementioned machine learning algorithms; and fast identification of the network status using one of the said models.

During the implementation of this system several challenges were met. The first consisted in obtaining a ground-truth scenario, done by simulating user behavior using one AP, four laptops and various typical traffic-generating applications. The second consisted in creating three types of artificial anomalies, which was done by: creating radio interference near the AP, enclosing it in reflective material (aluminum) and lowering its antenna signal power. A third challenge was performing feature reduction, which was achieved with a combination of the Extra-trees algorithm, Principal Component Analysis and network-related knowledge. Lastly it was also needed to choose the best algorithms, which was done by researching the state-of-the-art on the subjects of machine learning and testing them on sample data.

During the validation of the system’s results, high performance was achieved with the initial training and data set, with the system being able to correctly identify the network status in approximately three minutes, and the best results being achieved by the SVM model. The same models were used with new data two months later, and the performance dropped significantly. It was discovered that by applying the PCA technique to the new data, using principal components derived from the old data, allowed the Support Vector Machine model to achieve an F1-score of 0.69. However, these results were much worse when compared to those achieved by recreating the models with the complete new data, which were once again near perfect.

6.2 System Limitations and Future Work

Given the excellent results achieved with the ML models for SNMP data close in time to the identifications it would be interesting to create less intense anomalies, for example by setting the power of the
antenna to a stronger signal than the one used in here, covering the AP with a less reflective material or making the radio interference intermittent. Along with this, other types of anomalies could also contribute to scenarios which more closely resemble the real one, and thus the performance would be expected to drop. Examples of new anomalies could be using APs with known defective components, malicious attacks that lower the network’s quality of experience, or overloading the APs with too many connected users generating traffic.

Another limitation of this system is that it loses performance with time, as the network paradigm changes, and none of the known solutions are perfect. One of them would be to automatize the adaptation of the ML model to the new scenario by applying PCA to the new data, experimenting with varying numbers of principal components and choosing the one which provides the best results. However, while still better than random guessing, this process does not reach the performance of the other remaining solution, which consists in retraining the model with more recent data. The problem with this last alternative is that it requires recreating the artificial anomalies, and so it would not be automatic.

Another way to further explore the problem of network anomaly detection through machine learning techniques would be to combine the data from different APs closely placed, as many combined shifts in the values of SNMP parameters can indicate a problem with one of the APs: for example, a sudden drop in the number of users connected to one (problematic) AP and a following rise in other (normal) AP.


69


SNMP Parameters

Table A.1: List of the collected SNMP parameters, being the ones included in the final set in bold.

<table>
<thead>
<tr>
<th>OID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.3.6.1.2.1.1.3</td>
<td>&quot;The time (in hundredths of a second) since the network management portion of the system was last re-initialized.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.2.1.7.1</td>
<td>&quot;The total number of UDP datagrams delivered to UDP users.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.2.1.7.2</td>
<td>&quot;The total number of received UDP datagrams for which there was no application at the destination port.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.2.1.7.3</td>
<td>&quot;The number of received UDP datagrams that could not be delivered for reasons other than the lack of an application at the destination port.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.2.1.7.4</td>
<td>&quot;The total number of UDP datagrams sent from this entity.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.109.1.1.1.10</td>
<td>&quot;The overall CPU busy percentage in the last cpmCPUMonInterval period.&quot;</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>First entry</th>
<th>Second entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.3.6.1.2.1.2.2.1.10</td>
<td>“The total number of octets received on the interface, including framing characters.”</td>
</tr>
<tr>
<td>1.3.6.1.2.1.2.2.1.11</td>
<td>“The number of packets, delivered by this sub-layer to a higher (sub-)layer, which were not addressed to a multicast or broadcast address at this sub-layer.”</td>
</tr>
<tr>
<td>1.3.6.1.2.1.2.2.1.13</td>
<td>“The number of inbound packets which were chosen to be discarded even though no errors had been detected to prevent their being deliverable to a higher-layer protocol. One possible reason for discarding such a packet could be to free up buffer space.”</td>
</tr>
<tr>
<td>1.3.6.1.2.1.2.2.1.14</td>
<td>“For packet-oriented interfaces, the number of inbound packets that contained errors preventing them from being deliverable to a higher-layer protocol. For character-oriented or fixed-length interfaces, the number of inbound transmission units that contained errors preventing them from being deliverable to a higher-layer protocol.”</td>
</tr>
<tr>
<td>1.3.6.1.2.1.2.2.1.15</td>
<td>“For packet-oriented interfaces, the number of packets received via the interface which were discarded because of an unknown or unsupported protocol. For character-oriented or fixed-length interfaces that support protocol multiplexing the number of transmission units received via the interface which were discarded because of an unknown or unsupported protocol. For any interface that does not support protocol multiplexing, this counter will always be 0.”</td>
</tr>
<tr>
<td>1.3.6.1.2.1.2.2.1.16</td>
<td>“The total number of octets transmitted out of the interface, including framing characters.”</td>
</tr>
<tr>
<td>1.3.6.1.2.1.2.2.1.17</td>
<td>“The total number of packets that higher-level protocols requested be transmitted, and which were not addressed to a multicast or broadcast address at this sub-layer, including those that were discarded or not sent.”</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>First entry</th>
<th>Second entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.3.6.1.2.1.2.2.1.19</td>
<td>&quot;The number of outbound packets which were chosen to be discarded even though no errors had been detected to prevent their being transmitted. One possible reason for discarding such a packet could be to free up buffer space.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.2.1.2.2.1.20</td>
<td>&quot;For packet-oriented interfaces, the number of outbound packets that could not be transmitted because of errors. For character-oriented or fixed-length interfaces, the number of outbound transmission units that could not be transmitted because of errors.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.48.1.1.1.5</td>
<td>&quot;Indicates the number of bytes from the memory pool that are currently in use by applications on the managed device.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.48.1.1.1.6</td>
<td>&quot;Indicates the number of bytes from the memory pool that are currently unused on the managed device.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.48.1.1.1.7</td>
<td>&quot;Indicates the largest number of contiguous bytes from the memory pool that are currently unused on the managed device.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.10.1.1.3.1.1.4</td>
<td>&quot;This object will provide a cumulative count (since last system boot up or initialization) of the number of write retries that were done in the chip.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.10.1.1.3.1.1.5.1.1</td>
<td>&quot;This object will provide a cumulative count (since last system boot up or initialization) of the number of erase retries that were done in the chip.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.273.1.1.2.1.1</td>
<td>&quot;This is the number of wireless clients currently associating with this device on this interface.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.273.1.1.2.1.2</td>
<td>&quot;This is the number of bridges currently associating with this device on this interface.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.273.1.1.2.1.3</td>
<td>&quot;This is the number of repeaters currently associating with this device on this interface.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.273.1.1.3.1.1</td>
<td>&quot;This object counts the number of stations associated with this device on this interface since device re-started.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.273.1.1.3.1.2</td>
<td>&quot;This object counts the number of stations authenticated with this device on this interface since device re-started.&quot;</td>
</tr>
<tr>
<td>First entry</td>
<td>Second entry</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.273.1.1.3.1.3</td>
<td>&quot;This object counts the number of stations roamed from another device to this device on this interface since device re-started.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.273.1.1.3.1.4</td>
<td>&quot;This object counts the number of stations roamed away from this device on this interface since device re-started.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.273.1.1.3.1.5</td>
<td>&quot;This object counts the number of stations deauthenticated with this device on this interface since device re-started.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.273.1.1.3.1.6</td>
<td>&quot;This object counts the number of stations disassociated with this device on this interface since device re-started.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.273.1.1.4.1.2</td>
<td>&quot;This object counts the number of MIC failures encountered on this radio interface since device re-started.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.273.1.1.4.1.3</td>
<td>&quot;This object counts the number of MIC failures reported by clients on this radio interface since device re-started.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.273.1.1.4.1.4</td>
<td>&quot;This object counts the number of TKIP Counter Measures invoked on this interface since device re-started.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.273.1.1.4.1.5</td>
<td>&quot;This object counts the number of received unicast fragments discarded by replay mechanism on this interface since device re-started.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.273.1.1.4.1.6</td>
<td>&quot;This object counts the number of TKIP replay errors detected on this interface since device re-started.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.272.1.2.1.1.1.1</td>
<td>&quot;This counter shall increment when a frame transmission is deferred due to energy detection.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.272.1.2.1.1.1.2</td>
<td>&quot;This counter shall increment when a frame received has any MAC CRC error.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.272.1.2.1.1.1.3</td>
<td>&quot;This counter shall increment when a beacon or probe response frame received for which the SSIDs in the frame do not match the IEEE802dot11-MIB dot11DesiredSSID object.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.41.1.1.1</td>
<td>&quot;The number of clogMessageGenerated notifications that have been sent. This number may include notifications that were prevented from being transmitted due to reasons such as resource limitations and/or non-connectivity.&quot;</td>
</tr>
</tbody>
</table>

*Continued on next page*
<table>
<thead>
<tr>
<th>First entry</th>
<th>Second entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.3.6.1.4.1.9.9.41.1.1.4</td>
<td>&quot;The number of syslog messages which were ignored. A message will be ignored if it has a severity value greater than clogMaxSeverity.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.41.1.1.5</td>
<td>&quot;The number of syslog messages which could not be processed due to lack of system resources. Most likely this will occur at the same time that syslog messages are generated to indicate this lack of resources.&quot;</td>
</tr>
<tr>
<td>1.2.840.10036.1.1.1.15</td>
<td>&quot;This attribute holds the most recently transmitted Reason Code in a Disassociation frame. If no Disassociation frame has been transmitted, the value of this attribute shall be 0.&quot;</td>
</tr>
<tr>
<td>1.2.840.10036.1.1.1.17</td>
<td>&quot;This attribute holds the most recently transmitted Reason Code in a Deauthentication frame. If no Deauthentication frame has been transmitted, the value of this attribute shall be 0.&quot;</td>
</tr>
<tr>
<td>1.2.840.10036.1.1.1.19</td>
<td>&quot;This attribute holds the most recently transmitted Status Code in a failed Authentication frame. If no failed Authentication frame has been transmitted, the value of this attribute shall be 0.&quot;</td>
</tr>
<tr>
<td>1.2.840.10036.1.5.1.5</td>
<td>&quot;This counter shall increment when a frame is received with the WEP subfield of the Frame Control field set to one and the value of the ICV as received in the frame does not match the ICV value that is calculated for the contents of the received frame.&quot;</td>
</tr>
<tr>
<td>1.2.840.10036.1.5.1.6</td>
<td>&quot;This counter shall increment when a frame is received with the WEP subfield of the Frame Control field set to zero and the value of dot11ExcludeUnencrypted causes that frame to be discarded.&quot;</td>
</tr>
<tr>
<td>1.2.840.10036.2.2.1.1</td>
<td>&quot;This counter shall be incremented for an acknowledged MPDU with an individual address in the address 1 field or an MPDU with a multicast address in the address 1 field of type Data or Management.&quot;</td>
</tr>
</tbody>
</table>
### Table A.1 – Continued from previous page

<table>
<thead>
<tr>
<th>First entry</th>
<th>Second entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2.840.10036.2.2.1.2</td>
<td>“This counter shall increment only when the multicast bit is set in the destination MAC address of a successfully transmitted MSDU. When operating as a STA in an ESS, where these frames are directed to the AP, this implies having received an acknowledgment to all associated MPDUs.”</td>
</tr>
<tr>
<td>1.2.840.10036.2.2.1.3</td>
<td>“This counter shall increment when an MSDU is not transmitted successfully due to the number of transmit attempts exceeding either the dot11ShortRetryLimit or dot11LongRetryLimit.”</td>
</tr>
<tr>
<td>1.2.840.10036.2.2.1.4</td>
<td>“This counter shall increment when an MSDU is successfully transmitted after one or more retransmissions.”</td>
</tr>
<tr>
<td>1.2.840.10036.2.2.1.5</td>
<td>“This counter shall increment when an MSDU is successfully transmitted after more than one retransmission.”</td>
</tr>
<tr>
<td>1.2.840.10036.2.2.1.6</td>
<td>“This counter shall increment when a frame is received that the Sequence Control field indicates is a duplicate.”</td>
</tr>
<tr>
<td>1.2.840.10036.2.2.1.7</td>
<td>“This counter shall increment when a CTS is received in response to an RTS.”</td>
</tr>
<tr>
<td>1.2.840.10036.2.2.1.8</td>
<td>“This counter shall increment when a CTS is not received in response to an RTS.”</td>
</tr>
<tr>
<td>1.2.840.10036.2.2.1.9</td>
<td>“This counter shall increment when an ACK is not received when expected.”</td>
</tr>
<tr>
<td>1.2.840.10036.2.2.1.10</td>
<td>“This counter shall be incremented for each successfully received MPDU of type Data or Management.”</td>
</tr>
<tr>
<td>1.2.840.10036.2.2.1.11</td>
<td>“This counter shall increment when a MSDU is received with the multicast bit set in the destination MAC address.”</td>
</tr>
<tr>
<td>1.2.840.10036.2.2.1.12</td>
<td>“This counter shall increment when an FCS error is detected in a received MPDU.”</td>
</tr>
<tr>
<td>1.2.840.10036.2.2.1.13</td>
<td>“This counter shall increment for each successfully transmitted MSDU.”</td>
</tr>
</tbody>
</table>
Table A.1 – Continued from previous page

<table>
<thead>
<tr>
<th>First entry</th>
<th>Second entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2.840.10036.2.2.1.14</td>
<td>&quot;This counter shall increment when a frame is received with the WEP subfield of the Frame Control field set to one and the WEPOn value for the key mapped to the TA's MAC address indicates that the frame should not have been encrypted or that frame is discarded due to the receiving STA not implementing the privacy option.&quot;</td>
</tr>
</tbody>
</table>

Table A.2: List of the collected SNMP parameters which derived their values from the clients.

<table>
<thead>
<tr>
<th>OID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.3.6.1.4.1.9.9.273.1.3.1.1.3</td>
<td>&quot;This is a device-dependent measure of the signal strength of the most recently received packet from this client. It may be normalized or unnormalized.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.273.1.3.1.1.4</td>
<td>&quot;This is a device-dependent measure of the signal quality of the most recently received packet from this client.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.273.1.3.1.1.6</td>
<td>&quot;The number of packets received from this client.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.273.1.3.1.1.7</td>
<td>&quot;The number of bytes received from this client.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.273.1.3.1.1.8</td>
<td>&quot;The number of packets sent to this client.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.273.1.3.1.1.9</td>
<td>&quot;The number of bytes sent to this client.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.273.1.3.1.1.10</td>
<td>&quot;This counter increments when a packet for this client is received and the Sequence Control field in the packet header indicates the packet is a duplicate.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.273.1.3.1.1.11</td>
<td>&quot;This counter increments when an MSDU is successfully transmitted after one or more retransmissions for this client.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.273.1.3.1.1.12</td>
<td>&quot;This counter increments when an MSDU is not transmitted successfully for this client due to the number of transmit attempts exceeding some limit.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.273.1.3.1.1.13</td>
<td>&quot;This is the number of packets received from this client which failed to be properly decrypted via the WEP security mechanism.&quot;</td>
</tr>
<tr>
<td>1.3.6.1.4.1.9.9.273.1.3.1.1.14</td>
<td>&quot;This is the number of MIC errors for this client.&quot;</td>
</tr>
</tbody>
</table>

Continued on next page
Table A.2 – *Continued from previous page*

<table>
<thead>
<tr>
<th>First entry</th>
<th>Second entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.3.6.1.4.1.9.9.273.1.3.1.1.15</td>
<td>“This is the number of missing MIC packets for this client.”</td>
</tr>
</tbody>
</table>