Università degli Studi di Salerno
Dipartimento di Informatica

Dottorato di Ricerca in Informatica - XIII Ciclo

Tesi di Dottorato

Improving Network Anomaly Detection With Independent Component Analysis

Ugo Fiore

Anno Accademico 2013-2014

Candidato:	Coordinatore:	Tutor:
Ugo Fiore	Prof. Giuseppe Persiano	Prof. Alfredo De Santis

To the memory of Antonio Di Meo

Abstract

Complexity, sophistication, and rate of growth of modern networks, coupled with the depth, continuity, and pervasiveness of their role in our everyday lives, stress the importance of identifying potential misuse or threats that could undermine regular operation. To ensure an adequate and prompt reaction, anomalies in network traffic should be detected, classified, and identified as quickly and correctly as possible. Several approaches focus on inspecting the content of packets traveling through the network, while other techniques aim at detecting suspicious activity by measuring the network state and comparing it with an expected baseline. Formalizing a model for normal behavior requires the collection and analysis of traffic, in order to isolate a set of features capable of describing traffic completely and in a compact way. The main focus of this dissertation is the quest for good representations for network traffic, representation that are abstract and can capture and describe much of the intricate structure of observed data in a simple manner. In this way, some of the hidden factors and variables governing the traffic data generation process can be unveiled and disentangled and anomalous events can be spotted more reliably. We adopted several methods to achieve such simpler representations, including Independent Component Analysis and deep learning architectures. Machine learning techniques have been used for verifying the improvement in classification effectiveness that can be achieved with the proposed representations.

Acknowledgements

I would like to express my gratitude to my supervisor Alfredo De Santis, who gave me the opportunity of taking this PhD and is an endless source of inspiration, insights, and guidance.

I am indebted with the course coordinator Pino Persiano for creating and maintaining the conditions for fruitful research. Special thanks go to Enzo Loia and Paolo Zanetti for providing many stimulating opportunities. I would also like to thank Guglielmo Tamburrini, who gracefully accepted my decision to dedicate all my time to research.

I am grateful to Francesco Palmieri for his long-lasting friendship and collaboration, and to Nello Castiglione for his initiative and teamwork spirit.

Finally, my thanks from the deepest of my heart go to my wife, my daughter, and my son, who brighten every day of my life.

Contents

List of Figures

Chapter 1

Introduction

In the Internet economy, attacks are a growing concern. For example, distributed cooperative large-scale attacks such as Distributed Denial of Service (DDoS) attacks can have an impact at national level, involving entire countries. Identification and prevention of network abuses is thus becoming strategic to ensure an adequate protection from threats.

The network anomaly detection problem consists in analyzing the raw traffic transmitted over a network segment to determine if anything suspect or unusual is happening. The key assumption is that attacks patterns are fundamentally different from normal usage. On the other hand, anomalies in traffic need not necessarily reflect hostile activity: conditions of overload or particular events may cause significant deviations in the traffic patterns. For example, transient malfunctions in the network elements may induce congestion or broadcast storms. Evidently, the problem of anomaly detection in any complex system relies on the definition of the notion of what is ordinary, i.e., "normal". Normal behavior can be defined as what fits a model describing the traffic as the result of relations between all the fundamental components involved in the traffic dynamics, depending on several considerations and elements associated to the daily human activities accomplished by using the network. These include the applications running on the involved hosts, and the data they process. The main goal of anomaly detection is to devise techniques, typ-

ically based on machine learning, data mining or statistical analysis, that are able to model what a normal operating network should look like and, thus, detect and report any pattern that does not conform to such normal (also known as *baseline*) behavior. When considering network traffic flowing across one or more observation points, an event can be classified as anomalous if its degree of deviation from the "normal" network behavior observed at the same observation points is high enough. Thus, anomaly detection can also be viewed as a classification task. In the ideal case, all the data used for training belong to a single class (the "normal" traffic class) and a classifier has to learn the characteristics of such class and determine if any unseen instance belongs to it or not. Note also that the problem does not reduce to simply distinguishing true structure from noise in the data. Complexity is *inherent* in the interaction of the unknown explanatory factors, which can evolve over time at different speeds, can have both a local and a non-local span, and can have intertwined effects.

In anomaly detection and in other applications, the classification performance of many machine learning algorithms depends on the representation of data in terms of features. Effective machine learning techniques require a representation that can condense salient characteristics of data and facilitate combined analysis across features. Long and complex preprocessing pipelines are thus commonly designed and refined by human experts in order to achieve such representations. While this strategy yielded good results, it would be desirable that the effort of manual feature handcrafting were coupled with a form of extraction of relevant information from data where the focus changes: instead of leveraging domain-specific knowledge, one concentrates towards the discovery of unexpected features and correlations. In this perspective, it can be expected that "good" representations are those that abstract from the nitty-gritty details of low-level data and get closer to the latent explanatory factors and, ultimately, to the (unknown) underlying dynamics governing the generation of data.

One approach involves explicitly transforming the data in order to view it from a different standpoint, attempting at combining information from different time scales

or different observation points. Thus, transformations that get closer to the space of the explanatory variables can disentangle these variables. Moreover, transformations can also be useful in detecting invariant features that can be discarded in order to accelerate the analysis. In turn, this can help in reducing the number of dimensions, i.e., the number of features considered. The "curse of dimensionality", in fact, is a major problem, because it entails a combinatorial explosion in the number of cases to be scrutinized. Reducing the number of dimensions to analyze, by selecting the ones that convey the most information, is an important step towards an efficient classification.

This question is also related to the circumstance of learning being supervised or unsupervised. In a supervised setting, i.e., with data associated to labels indicating if an instance is normal or anomalous, "good" features are those that, separately or in combination, have a high classification power. Labeled data, however, are hardly available in large quantities, sufficient to encompass a satisfactorily wide variety of cases that can be considered as fully representative of the population. Specifically, in network anomaly detection, labeling data is a lengthy and costly operation, because it involves considerable manual intervention by experts. This is why research has somewhat shifted its objectives to the unsupervised or semi-supervised settings.

A natural question that raises is about which transformations produce a positive impact on discovering unknown structure in data. Some transformations could hide such structure, bringing confusion instead of clarity. Informally, one can say that representations which are appropriate will make modeling the distribution of data more straightforward than will be the case with inappropriate representations. Intuitively, if a representation can be found that simplifies the model, that would mean that at least some of the explanatory factors have been isolated.

Another approach that is actively explored is the devising of generative models that, without human intervention, produce an abstract representation of the observed data. Much work in the deep learning research community [1] is focused on obtaining such abstract representations by means of stacking layers of auto-encoders or Restricted Boltzmann Machines (RBMs) in such a way that only the lowest layer

receives the original data as input, while each upper layer produces a representation of the output of the layer below it [2]. In this way, progressively more abstract representations can be obtained, establishing nonlinear mappings between data and features and between low-level features and high-level ones. It is worth noting that many classical techniques can be reduced to special cases of the general graphical model [3].

Although deep learning algorithms can separate out the explanatory factors to some extent, relying exclusively on them might not be the ideal avenue to a successful solution of the network anomaly detection problem. Firstly, there is no guarantee that the abstractions found would reflect all the governing variables. Secondly, increased complexity can slow down the learning process. Thirdly, it is difficult to have a deep model explain its decisions. The abstract representations yielded by deep learning algorithm can, however, be expected to elucidate, at least partially, the dynamics of the observed system by exposing some of the latent variables that are responsible for its evolution. It is natural, then, to think of complementing the disentangling ability of deep learning algorithms with other ways of obtaining representations that capture relevant structure that may not be immediately evident in the data distribution.

1.1 Our contribution

The main goal of this dissertation is to delineate an approach to network anomaly detection based on the search of representations that simplify the model of network traffic. Two strategies to attain such representations are described. The first involves the use of artificial intelligence techniques to find the appropriate representation in a fully automated, unassisted way. The second strategy capitalizes upon methods from signal analysis, discussing the effects of a transformation of input that separates the data into independent components.

In particular, a first study involved the analysis of traffic and the construction of an automated representation by means of a generative model, the RBM. The

approach based on RBMs lays down the framework for automated generation of representation through deep, layered, architectures, where each layer generates its "portray" of the data it receives from lower layers. Upper layers produce an abstract representation that can expose some of the latent variables, though the abstraction is not easy to visualize and analyze. The emphasis is on a semi-supervised setting, where only limited labeled data is available to train the classifier.

Subsequently, separation of the characteristics of the observed data in the Independent Components was applied as a different way to extract relevant characteristics from observations. The procedure employing Independent Component Analysis (ICA) is founded on the basic intuition that observable traffic is the superposition of individual streams, each associated to a specific phenomenon or to a particular category of traffic. Separating such streams has the potential to expose variations with respect to the normal behavior in more efficient way than it is achievable by examining raw, untransformed traffic only. Accordingly, we looked into the traffic time series by inspecting the independent components responsible for the evolution of traffic and evaluating their variation with respect to the baseline. We also tested the effectiveness of separation in improving the classification accuracy of classifiers.

A trait which is shared by both the techniques proposed is that the intrinsic components obtained make discernible some information that is not immediately evident with other statistic features calculated directly on the observed data. The derived representations can disentangle the latent factors that govern the generation of observable data, help build a simpler model and ultimately give insights for better understanding the data-producing process. We provide a framework that could also be applied to anomaly detection problems in similar domains, because its flexibility makes it adaptable in a wide range of contexts, especially when available data is collected over multiple sampling points.

1.2 State of the art

Several works addressed the network anomaly detection problem by analyzing the traffic time series in order to isolate deviations. Several surveys and books have been published on the topic of network anomaly detection, and a recent survey article also contains a comparison of prior surveys [4]. A wide range of methods have been used, including exponential smoothing and forecasting based on the Holt-Winters decomposition [5], adaptive setting of thresholds and cumulative sum [6]. Initial attempts [7] analyzed audit trail records and underlined the importance of scrutinizing activity coming from inside as well as from outside. As research reached more maturity, in both theory and methods, general techniques were adjoined by methods aimed specifically at a single threat (e.g., DDoS or botnet detection [8]), or a particular network domain (e.g., Wireless Sensor Networks (WSNs)).

One of the first contributions to convey the idea that the observed data were the combined effect of multiple separated sources was the work by Lakhina *et al.* [9]. They measured the traffic on backbone network links and considered that aggregate traffic counters as the linear sum of the contribution of individual Origin-Destination (OD) flows. The separation technique they applied involved only second-order statistics. The survey of Chandola *et al.* [10] introduced the notion of point, contextual, and collective anomalies. Point anomalies differ from expected patterns as single observations, anomalies of the second kind are anomalous when referenced to a particular context, while collective anomalies involve a collection of events that may not constitute anomalies when considered individually.

In the literature, there are various machine learning approaches for intrusion detection systems, such as neural networks (NNs), fuzzy logic (FL), genetic algorithms (GAs), genetic programming (GP), swarm intelligence (SI) and artificial immune systems (AISs). Recently, techniques of soft computing were also applied to network anomaly detection. Wu and Banzhaf surveyed the approaches to intrusion detection based on computational intelligence [11]. Metrics originating from information theory were studied in anomaly detection by Lee and Xiang [12]. Entropy-based

detection systems are promising for the specific individuation of DDoS attacks, in part because they are robust with respect to changes in network utilization. Shannons entropy and Kullback-Leibler divergence methods are assumed to be the most effective methods. An empirical evaluation of information metrics can be found in [13].

The particular characteristics of network traffic when observed on multiple time scales, with long-ange dependencies (LRD) and heavy-tailed distributions stimulated works studying the spectral analysis and autocorrelation, as well as wavelet analysis [14]. Techniques such as Recurrence Quantification Analysis (RQA) were first applied to network anomaly detection in our previous work [15]. Reconstruction of the unknown dynamics of the traffic data generating process was attempted through delay-coordinate embedding in conjunction with Average Mutual Information and False Nearest Neighbors, and RQA was then used to isolate unusual patterns.

1.3 Organization of this thesis

The next chapter provides motivation for the network anomaly detection problem, framing it into a description of proposed solutions. In the third chapter, the ideas and motivations of looking for appropriate representations are introduced, following with a discussion of deep learning architectures and undirected graphical models. The complex structure of network traffic suggests that automated methods to change the feature space of data and attaining more abstract representation can help producing a simplified model, where normal and abnormal events are clearly separated. Explicit transformations, by means of methods from signal processing, are a different, but not necessarily alternative strategy to get closer to the underlying variables governing the traffic production process. These aspect are discussed in detail in the fourth chapter. The fifth chapter reports together with an overview of the techniques to assess the performance of classifiers aimed at separate anomalous events from normal traffic. In the sixth chapter, a proof of concept implementation and experiments evaluating the effects of applying the separation of independent

components to the detection of volume-based attacks are described. Validation has been carried out by using real traffic data from a production network. In the final chapter, after drawing some conclusions, directions for further research are sketched, including some ongoing efforts and ideas to be explored later on.

References

[1] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: a review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[2] G. E. Hinton, "Learning multiple layers of representation," *Trends in cognitive sciences*, vol. 11, no. 10, pp. 428–434, 2007.

[3] A. Cichocki and S.-i. Amari, *Adaptive blind signal and image processing: learning algorithms and applications*. John Wiley & Sons, 2002.

[4] M. Bhuyan, D. Bhattacharyya, and J. Kalita, "Network anomaly detection: methods, systems and tools," *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, pp. 303–336, 2014.

[5] J. Brutlag, "Aberrant behavior detection in time series for network monitoring," in *Proceedings of the 14th USENIX conference on System administration*, 2000, pp. 139–146.

[6] V. Siris and F. Papagalou, "Application of anomaly detection algorithms for detecting SYN flooding attacks," *Computer Communications*, vol. 29, no. 9, pp. 1433–1442, 2006.

[7] J. P. Anderson, "Computer Security Threat Monitoring and Surveillance," Computer Security Division of the Information Technology Laboratory, National Institute of Standards and Technology, Fort Washington, Pennsylvania, Tech. Rep., 1980.

[8] G. Gu, R. Perdisci, J. Zhang, W. Lee, *et al.*, "Botminer: clustering analysis of network traffic for protocol-and structure-independent botnet detection.," in *USENIX Security Symposium*, 2008, pp. 139–154.

[9] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," in *ACM SIGCOMM Computer Communication Review*, ACM, vol. 34, 2004, pp. 219–230.

[10] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: a survey," *ACM Comput. Surv.*, vol. 41, no. 3, 2009.

[11] S. X. Wu and W. Banzhaf, "The use of computational intelligence in intrusion detection systems: a review," *Applied Soft Computing*, vol. 10, no. 1, pp. 1–35, 2010.

[12] W. Lee and D. Xiang, "Information-theoretic measures for anomaly detection," in *Proceedings, 2001 IEEE Symposium on Security and Privacy (S&P 2001)*, IEEE, 2001, pp. 130–143.

[13] M. H. Bhuyan, D. Bhattacharyya, and J. Kalita, "An empirical evaluation of information metrics for low-rate and high-rate ddos attack detection," *Pattern Recognition Letters*, vol. 51, pp. 1–7, 2015.

[14] X. Tian, H. Wu, and C. Ji, "A unified framework for understanding network traffic using independent wavelet models," in *Proceedings, Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. INFOCOM 2002*, IEEE, vol. 1, 2002, pp. 446–454.

[15] F. Palmieri and U. Fiore, "Network anomaly detection through nonlinear analysis," *Computers & Security*, vol. 29, no. 7, pp. 737–755, 2010.

Chapter 2

Network anomaly detection

Anomaly detection in network traffic has been studied at least since 1980, when Anderson, analyzing system audits, characterized threats originated by external penetrators or internal misfeasors and suggested that audit trails could point to misbehavior [1]. In the work of García-Teodoro *et al.* [2], anomaly-based Network Intrusion Detections Systems (A-NIDS) were classified into three main groups, according as their basis: knowledge, statistical, and machine learning.

Knowledge-based anomaly detection systems attempt at classifying data from auditing and other sources in accordance with a set of rules that describe legitimate behavior. Rules are built—and maintained—by incorporating human expert knowledge. Often these rules are protocol-based, and therefore they address only a specific portion of the problem. The process of setting up the rules, detecting and correcting errors with them, and preserving their efficacy over time is lengthy and requires extensive human intervention. Consequently, it results in high costs.

Statistical-based anomaly detection systems build a statistical profile characterizing the distribution of observed data, and then compare the identified behavior to the profile associated with normal behavior, flagging as anomalous the events that do not conform, or conform badly, to the normal profile. In statistical models, the usual assumption is that observations are independent and independently distributed (i.i.d.) samples taken from an unknown probability distribution $\mathcal{P}$. Parametric methods assume that the form of the distribution is known and only its parameters have to be estimated given the observations, for example by maximum likelihood.

When nothing can be assumed about the structure of the data-generating distribution $\mathcal{P}$, non-parametric methods such as histograms are used to gain insights into the distribution. Frequency thresholds can then be applied to identify anomalies. However, histogram-based methods fall short of describing the interactions among different features.

Machine learning-based anomaly detection systems build an implicit or explicit model to represent normal behavior and categorize observed patterns with reference to that model. A decision boundary is then sought to separate normal and anomalous data points. Such decision boundary is learnt from training data. The detection system is thus trained on behavioral characteristics of network traffic, and it triggers alerts when it finds irregular traffic patterns.

Based on the nature of input data to be analyzed, anomaly detection systems can also be divided in packet-based, flow-based, and window-based [3]. Raw packets are inspected and evaluated individually in packet-based systems. This causes scalability problems, if the explosive growth in the network throughput is considered. Aggregated information can provide sufficient detail for the analysis while at the same time maintaining a granularity suitable for practical collection and analysis. Aggregation can be done at the flow level or by using time windows. Packets with the same source address and port, destination address and port, and protocol belong to the same flow. Window-based systems, instead, group input data into sliding time windows and analyze the evolution of patterns over time windows.

A traditional taxonomy of network intrusion detection systems is based on the strategy used to flag significant events. *Misuse-detection* systems rely on precise descriptions of what constitutes malicious behavior, whereas *anomaly-detection* systems characterize normal activity in some way and identify deviations from the baseline [4]. It is worth noting that the above-specified grouping bears some similarity with the subdivision of machine learning contexts in the supervised setting, on one hand, and the semi-supervised and unsupervised settings, on the other. Although misuse detectors have analyzed various sources of data, including logs or system call traces [2], the most common form of such detectors is based on signature analysis.

Signature-based systems scan network traffic searching for byte sequences that are specifically associated to malicious activity. While such systems can quickly spot many attacks, their main drawback is that they rely on prior knowledge and can only cope with attacks whose signature has been isolated and exposed. In other words, these systems are limited by the coverage and timeliness of their database of signatures. They are thus vulnerable to zero-day, polymorph, and evolving attacks.

Anomaly detection, by contrast, relies on the hypothesis that attacks patterns have characteristics that constitute a deviation from normal behavior. Generally speaking, anomaly detection is not affected by the shortcomings of misuse detection. In theory, new attacks and variations of existent attacks can be detected. In practice, however, several difficulties arise.

2.1 What makes network anomaly detection hard

Sommer and Paxson [4] delineated several respects in which network anomaly detection is different—and fundamentally harder—from other applications of machine learning. After arguing that the negative definition of "anomalous" (the word *anomalous* actually means *not* normal) is challenging because recognizing similarity to previously seen examples is easier than recognizing deviations from what is expected, they list characteristics that significantly hinder application of machine learning to network anomaly detection. A straightforward approach to anomaly detection would define a region in some space and declare anomalous all the observations that lie outside that region. This procedure would require that the boundary between the normal region and the outside be sharp, increasing the difficulty of specifying the normal region. Even if a perfect system were able to recognize examples similar to what it had previously seen, discovering outliers by characterizing them as dissimilar from the known patterns would be difficult. The perfect system would need, in fact, to have a complete model of normality, where all possible cases are encompassed and described. Otherwise, every new form of regular behavior would

be classified as anomalous. The assumption that the ensemble of normal examples is wide enough to cover all possible variation is called the *closed world* assumption [5]. The provocative comment that closed the paper by Gates and Taylor [6] is still valid:

> Perhaps it is time that we applied anomaly detection to the detection of legitimate traffic, filtering it out and leaving the majority for further analysis

In intrusion detection, the cost of misclassification is remarkably high. False positives cause a significant waste of expert time to examine the reported attack only to conclude that there was no incident. Too many false positives would thus make an anomaly detection systems unusable. On the other hand, false negatives carry a cost that can be still more serious: the compromise of even a single system can jeopardize the integrity of the entire structure, and recovery could take extensive resources and time.

In order to appropriately train a detection system so that it will achieve a good ability to recognize normal examples, a dataset will be needed that contains no malicious patterns. However, it is hard to certify that a dataset is completely free from attacks. This conclusion is supported by precisely the same argument that underlines the advantage of anomaly detection with respect to signature-based analysis, i.e., that the latter cannot detect previously unseen attack patterns. A novel attack could be present in a dataset and nevertheless go undetected. In addition, all considerations about the costs of preparing a clean dataset beforehand apply.

Systems trained to discover deviations from normal behaviour unavoidably have to set a threshold beyond which an example is considered unexpected and hence anomalous. Once such a threshold has been established, attackers may configure their malicious activity in such a way that it would systematically stay under the threshold. For example, Özçelik and Brooks [7] have recently shown how entropy-based detectors for DDoS attacks can be deceived by spoofing addresses in such a way as to adjust the entropy of the header fields, making it stay in a predefined

range. This problem is exacerbated when thresholds are adaptively modified to reflect variations in normal activity, as attackers may adjust their endeavors to escape detection.

In addition to the themes raised in [4], we can add two further reasons why network anomaly detection faces additional difficulty with respect to other machine learning tasks. Once attackers arrive at knowing the anomaly detection mechanism that is in place, besides adapting the profile of their attack to make it stay just under the critical threshold, they also have another tactic available to them that can degrade the efficacy of the detection system. Attackers could fabricate single packets or sets of packets with the sole purpose of making the detection system issue false alarms [7]. These attacks would come from spoofed IP addresses and impact different targets from the real ones, because attackers would wish to preserve the machines they control (likely through bots) and simultaneously avoid to disclose the real objectives they are interested in.

Modeling network traffic is a satisfactory way is also harder than what can be intuitively anticipated. The traffic profiles may vary substantially from one network environment to another, due to several factors including the characteristics of users, the architecture of machines, and the operating system run on them. Furthermore, network administrators usually set and enforce traffic-regulating policies which specify traffic types that are allowed or privileged. In their 2001 analysis [8], Floyd and Paxson observed that simulating Internet traffic suffered from three major difficulties. The first one derives from design choices in the IP protocol, privileging connective ability over uniformity of behavior. The second one relates to the dimension and growth of the Internet, which complicate the setting of thresholds for "rare" events in terms of percentage of occurrence. The third one depends on the quick rate of change over time of the Internet and of its structure.

Even within a single network, the fluctuation of measured parameters can be extremely high, leading to bursty and heavy-tailed distributions. Evaluation of burstiness made on a single time scale can only account for spatial correlation between traffic counters. When the observation time scale gets longer, observations tend to

exhibit greater smoothness, but there are still intricate relationships between phenomena occurring at different time scales. To fully describe the effects of temporal correlation and LRD, an analysis spanning multiple time scales is required [9].

References

[1] J. P. Anderson, "Computer Security Threat Monitoring and Surveillance," Computer Security Division of the Information Technology Laboratory, National Institute of Standards and Technology, Fort Washington, Pennsylvania, Tech. Rep., 1980.

[2] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *Computers & Security*, vol. 28, no. 1-2, pp. 18–28, 2009.

[3] J. Wang, D. Rossell, C. G. Cassandras, and I. C. Paschalidis, "Network anomaly detection: a survey and comparative analysis of stochastic and deterministic methods," in *IEEE 52nd Annual Conference on Decision and Control (CDC)*, IEEE, 2013, pp. 182–187.

[4] R. Sommer and V. Paxson, "Outside the closed world: on using machine learning for network intrusion detection," in *2010 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2010, pp. 305–316.

[5] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[6] C. Gates and C. Taylor, "Challenging the anomaly detection paradigm: a provocative discussion," in *Proceedings of the 2006 workshop on New security paradigms*, ACM, 2006, pp. 21–29.

[7] İ. Özçelik and R. R. Brooks, "Deceiving entropy based dos detection," *Computers & Security*, 2014.

[8] S. Floyd and V. Paxson, "Difficulties in simulating the internet," *IEEE/ACM Transactions on Networking (TON)*, vol. 9, no. 4, pp. 392–403, 2001.

[9] X. Tian, H. Wu, and C. Ji, "A unified framework for understanding network traffic using independent wavelet models," in *Proceedings, Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. INFOCOM 2002*, IEEE, vol. 1, 2002, pp. 446–454.

Chapter 3

Finding Appropriate Representations

When modeling complex phenomena, a central challenge is to find a representation of data that can disentangle, at least to some extent, the unknown factors that are responsible for the variation of the observed data. A representation is satisfactory when it captures all essential aspects of the observed system, yet in a compact way. From the standpoint of supervised learning, for example, a good representation is given by features with the greatest predictive power; a subset of the features is able to account for a major part of the variation in the observations. In the unsupervised setting, instead, it is not obvious to state what constitutes an appropriate representation. Intuitively, if a representation can be found that produces a simplified model, that would mean that at least some of the intricately connected explanatory factors have been somewhat isolated.

This is closely related to the problem of reducing the dimensionality of data and, more generally, to the manifold hypothesis. Identifying and discarding irrelevant dimensions is known from a long time to be key to achieving a compact model and, in turn, efficient machine learning algorithms. The manifold hypothesis [1][2] states that, in many scenarios of practical interest, the probability mass for the data-generating distribution concentrates near a collection of m-dimensional manifolds projected in the n-dimensional input space (with $m < n$). Manifolds associated with different classes tend to be separated by regions characterized by low probability density. If spaces can be found in which the manifolds have a simpler structure than their projections, modeling and classification would be greatly simplified in

these spaces. The ultimate goal is to find a representation in which the classes are linearly separable, i.e., they can be divided by a set of hyperplanes. Efficient methods such as Support Vector Machines are available for linearly separable data. A supporting argument in favor of the manifold hypothesis is that, if the hypothesis were not true, then synthesizing data by sampling randomly over observable features would produce reasonably-looking data much more often than it is empirically seen.

The manifold hypothesis also suggests interesting connections to the *phase space attractors* of chaos theory [3]. Assuming that the observed data is the visible part of a dynamical system that is governed by some differential equations (whose number and form are both unknown), then the system state is determined by a number of latent variables. Observable features are the result of (typically complex) interactions between entangled state variables. Notably, since the state variables can interact in many ways, the number of observed variables can be higher than the number of state variables. Few latent variables can therefore produce very complex structures. Given the observations, reconstructing the space of the latent state variables (the *phase space*) is challenging, but can provide deep insights into the underlying dynamics governing the system. Considering a single observed variable, by means of such techniques as delay-coordinate embedding, the "natural" dimension in which to describe the observations can be found. Delay-coordinate embedding reconstructs the phase space starting from a time series, by assembling observations taken at regular intervals into vectors belonging to the phase space. In this process, both the dimension of the phase space and the time delay for embedding should be estimated. In particular, autocorrelation or mutual information can reveal some periodic structure in the time series, which can hint at good values to be tried for the time delay. To estimate the appropriate embedding dimension, the method of False Nearest Neighbors (FNNs) is generally adopted . When the reconstructed dimension approaches the "correct" one, trajectories in phase space can be seen to "unfold": points that used to be close neighbors in up to k dimensions become far in $k+1$ dimensions. Proximity can, in fact, derive from the effects of projection, with no relation to the actual closeness of points. The percentage of FNNs can

be expected to be steady as the embedding dimension increases, because the errors induced by projection tend to be distributed uniformly. When the "natural" embedding dimension is reached, however, the percentage of FNNs should drop, as a consequence of the unfolding structure.

Once a good representation has been found, with probability mass concentrated in a predictable and regular way in the representation space, estimating the probability distribution of the data becomes easier. This point can be further illustrated if interpolation is considered. Usually, interpolating in the input space produces examples whose characteristics are very different from those of examples drawn from the real distribution. That behavior stems from the fact that the complexity of interactions produces involved topologies in the input space. Points that appear to be close in the input space can actually be far in representation spaces endowed with a different number of dimensions. In contrast, an ideal representation would "flatten" the manifold so that the transformed high-probability manifold becomes closer to a convex set [4], so that generating samples by interpolation in the representation space would yield reasonably-looking results. Experimentally, this flattening effect has been observed with deep learning architectures such as stacked denoising auto-encoders or RBMs [5][6].

Deep learning is concerned with learning multiple levels of representation, associated, in a sense, with different levels of abstraction relevant to the distribution of interest. Just as more sophisticated levels of abstraction are defined hierarchically as elaborations of the lower-level ones, deep learning architecture are built by stacking layers of neural networks in such a way that the output of each layer makes the input for the layer above it. Learning, in deep architectures, is difficult precisely for the same reason that makes deep architectures powerful, i.e., because of the growth in complexity of inter-layer dependencies as the number of layers increases. To cope with this problem, training signals should be propagated across the different layers in a controlled way. One of the techniques used to this purpose is greedy layer-wise pretraining: each layer is trained independently of the layers above it and pre-trained layers are then combined to form the final architecture.

Denoising auto-encoders (DAE) [7] are a class of greedily pre-trainable layers where noise is deliberately added to input (during training only) and the auto-encoder learns to reconstruct the original signal. Auto-encoders learn a pair of functions from input space to representation space (the encoding function) and vice versa (the decoding function) with the property that the composition of decoder and encoder (the *reconstruction*) is close to the original sample. Obviously, an auto-encoder might simply learn the identity function. This is prevented by regularization, determining low reconstruction errors for examples in the training set, but high reconstruction errors elsewhere. Note that the layer-wise pretraining is unsupervised because no label is necessary: the correct input is already available for comparison with the reconstruction. For an observed random variable X, denote the data-generating distribution with $\mathcal{P}(X)$. Then a known corruption process $\mathcal{C}$ will stochastically map X to $\tilde{X}$ according to the conditional distribution $\mathcal{C}(\tilde{X}|X)$. Now, to obtain the training data, a number of datapoints $(X, \tilde{X})$ are sampled from the data-generating process and the corruption process, i.e., $X \sim \mathcal{P}(X)$ and $\tilde{X} \sim \mathcal{C}(\tilde{X}|X)$. The denoising auto-encoder learns a conditional distribution $\mathcal{Q}(X|\tilde{X})$, that is to predict X given $\tilde{X}$ by regularized maximum likelihood.

3.1 Undirected models

A neural network may contain "inward" and "outward" connections between its layers. Outward connections, also commonly known as top-down or "generative" connections, go from the inner hidden layers to the outer hidden layers and from the outermost hidden layer to the visible neurons. These connections constitute the model the network has built of the training data. Weights on outward connections are modified, during learning, to maximize the probability that the generated patterns coincide with training data. Inward connections, commonly also known as bottom-up or "recognition" connections, go from the visible neurons to the outermost hidden layer and from outer layers to inner layers. These connections are responsible for explaining, in a sense, how the network could have generated a pat-

tern., although it is challenging to actually explain the behavior of a network in simple terms. Similarly, finding out why a neural network performs unsatisfactorily is not an easy task.

A Boltzmann machine (BM) is a network of stochastic neurons with bidirectional, symmetrical links [8]. In RBMs, the connections between layers constitute the totality of connections, as no intralayer connections are allowed; hence the *Restricted* term in RBM. Put differently, the units of a RBM are the vertices of a bipartite undirected graph. All the stochastic binary units (feature detectors) in the hidden layer are influenced by all the stochastic binary units in the visible layer, and vice versa. With respect to general BMs and directed graphical models, RBMs have the distinctive trait that inferring the state of hidden units, given the state of the visible units, is straightforward. Contrarily, in unrestricted BMs and directed models, inference is difficult.

In a RBM, the conditional distributions of the hidden $\mathbf{h}$ and visible $\mathbf{v}$ stochastic binary units, $p(\mathbf{h}|\mathbf{v})$ and $p(\mathbf{v}|\mathbf{h})$, factorize because there are no intralayer links. Considering a sigmoid linkage

$$p(\mathbf{h}|\mathbf{v}) = \prod_j p(h_j|\mathbf{v}) = \prod_j \text{sigm}(a_j + \sum_i W_{ij} v_i) \tag{3.1}$$

and similarly

$$p(\mathbf{v}|\mathbf{h}) = \prod_i p(v_i|\mathbf{h}) = \prod_i \text{sigm}(b_i + \sum_j W_{ij} h_j) \tag{3.2}$$

where

$$\text{sigm}(x) = \frac{1}{1 + \exp(-x)} \tag{3.3}$$

is the logistic sigmoid function, $\mathbf{a}$ and $\mathbf{b}$ are biases, and $\mathbf{W}$ is the matrix of weights. The model parameters are collectively denoted with $\theta = \{\mathbf{a}, \mathbf{b}, \mathbf{W}\}$. The logistic sigmoid is a commonly chosen function for coupling units (together with the tanh function, which is affine to it), because it resembles the shape of a step function, yet it is differentiable. RBMs can be readily extended to handle real-value visible units.

The probability of a joint configuration of the stochastic hidden and visible units can be related to a potential energy function of the states, in such a way that the

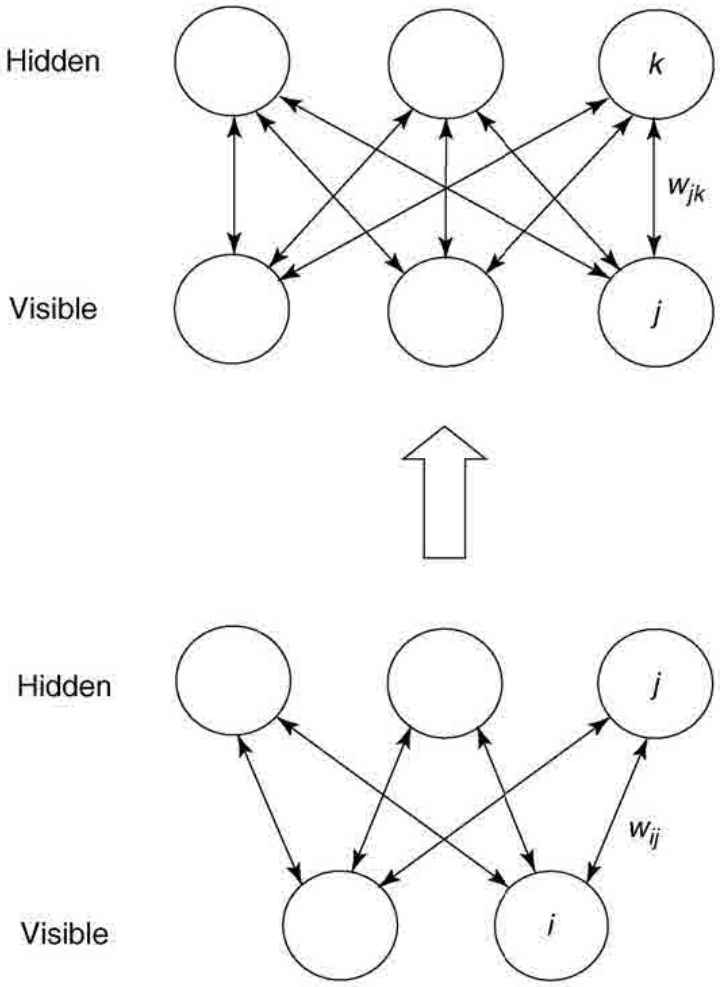

Figure 3.1: Two Restricted Boltzmann Machines stacked [9].

network will tend to stay in low-energy states. A joint configuration of the the hidden and visible stochastic binary units has an energy

$$\mathcal{E}(\mathbf{h}, \mathbf{v}) = -\mathbf{a}^T\mathbf{h} - \mathbf{b}^T\mathbf{v} - \mathbf{v}^T\mathbf{W}\mathbf{h} \tag{3.4}$$

and the probability of a joint configuration is proportional to e to the power of its energy:

$$p(\mathbf{h}, \mathbf{v}) \propto \exp(-\mathcal{E}(\mathbf{h}, \mathbf{v})) \quad . \tag{3.5}$$

To get the probability, a normalization factor is needed, namely the *partition func-*

tion[1]

$$Z = \sum_{\mathbf{v}} \sum_{\mathbf{h}} \exp(-\mathcal{E}(\mathbf{h}, \mathbf{v})) \tag{3.6}$$

thus obtaining the Boltzmann distribution

$$p(\mathbf{h}, \mathbf{v}) = \frac{\exp(-\mathcal{E}(\mathbf{h}, \mathbf{v}))}{Z} . \tag{3.7}$$

The partition function is intractable because it involves summing over all possible joint configurations.

Learning involves maximizing the average likelihood that the model would generate the training data. Marginalizing over all possible hidden vectors, the probability of a configuration of the visible units

$$p(\mathbf{v}) = \sum_{\mathbf{h}} p(\mathbf{h}, \mathbf{v}) = \frac{1}{Z} \sum_{\mathbf{h}} \exp(-\mathcal{E}(\mathbf{v}, \mathbf{h})) \tag{3.8}$$

is obtained. Defining the *free energy* $\mathcal{F}$ as

$$\mathcal{F}(\mathbf{v}) = -\log \sum_{\mathbf{h}} \exp(-\mathcal{E}(\mathbf{v}, \mathbf{h})) \tag{3.9}$$

it follows that

$$\sum_{\mathbf{h}} \exp(-\mathcal{E}(\mathbf{v}, \mathbf{h})) = \exp(-\mathcal{F}(\mathbf{v})) \tag{3.10}$$

and

$$Z = \sum_{\mathbf{v}} \exp(-\mathcal{F}(\mathbf{v})) . \tag{3.11}$$

Using Eq. (3.10), Eq. (3.8) can be written in a similar form to Eq. (3.7):

$$p(\mathbf{v}) = \frac{\exp(-\mathcal{F}(\mathbf{v}))}{Z} . \tag{3.12}$$

Considering the logarithm of $p(\mathbf{v})$

$$\log p(\mathbf{v}) = -\mathcal{F}(\mathbf{v}) - \log Z \tag{3.13}$$

[1]The partition function is usually denoted with the letter Z, from the German word *Zustandssumme*—sum over states

the derivative of the aggregate log likelihood (averaging over the training datapoints) with respect to the parameters can be written (details are in Appendix A) as

$$\frac{\partial}{\partial\theta}\log p(\mathbf{v}) = -E_{\mathcal{D}}\left\{\frac{\partial}{\partial\theta}\mathcal{E}(\mathbf{v},\mathbf{h})\right\} + E_{\mathcal{M}}\left\{\frac{\partial}{\partial\theta}\mathcal{E}(\mathbf{v},\mathbf{h})\right\} \tag{3.14}$$

that is, as the difference of two expectations. The first expectation, $E_{\mathcal{D}}\{\cdot\}$, is relative to the posterior distribution $\mathcal{D}$ of training data and is therefore data-dependent, the second expectation, $E_{\mathcal{M}}\{\cdot\}$, is relative to the posterior distribution $\mathcal{M}$ defined by the model and is therefore data-independent. Note that the hidden units are independent given a single data vector, but the aggregated posterior over all training vectors will not factorize. Exact maximum likelihood learning is intractable, because the data-independent expectation involves summation over all possible configurations. In a RBM, learning can be done through Gibbs sampling, alternatively in the inward and outward direction, until an equilibrium is reached so that the weights will not change. In detail, the data-independent part could be estimated by inferring the states of hidden units from training datapoints, then starting Markov chains at the inferred hidden states, updating alternatively all the visible units in parallel and all the hidden units in parallel, until convergence. In practice, efficient learning can be achieved with Contrastive Divergence [10], with a limited number of Gibbs sampling steps.

Deep architectures can be built by stacking RBMs on top of one another, as it is shown in Figg. 3.1 and 3.2. The activities of the hidden units of the "outside" RBM become the inputs for the "inside" RBM as in Fig. 3.1. In the resulting generative model, only connections between the topmost two layers are undirected; the other connections to units in the lower layers are directed (i.e., outward only) 3.2. This model is called Deep Belief Network (DBN). Learning, in such architectures, can be done in a layer-wise fashion. A proper Deep Boltzmann Machine can be attained by averaging the weights of the connections that a layer has with the upper and the lower layer next to it in the RBM stack and appropriately modifying the weights in the middle layers to avoid the effect of double-counting [11].

Another interesting related approach was laid out in reference [12], where Neigh-

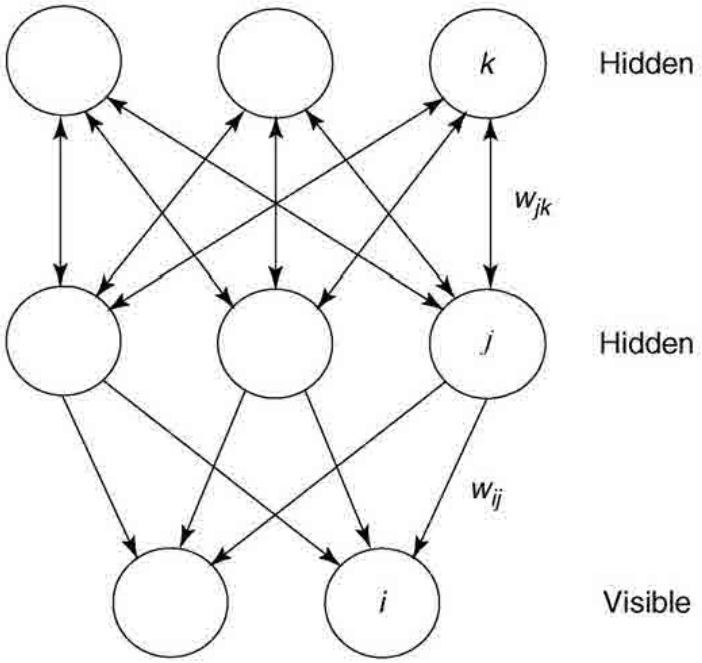

Figure 3.2: The composite generative model from the RBMs in Fig. 3.1 [9].

bourhood Components Analysis (NCA) is introduced. Starting from the properties of the k-Nearest-Neighbor (kNN) classifier, namely its nonlinear decision surfaces and low overfitting, Goldberger *et al.* estimate a linear transformation which maximizes the expected number of points correctly classified by kNN. A neural network to learn the parameters of a nonlinear transformation with the same objective was then proposed [13].

References

[1] L. Cayton, "Algorithms for manifold learning," UCSD, Tech. Rep., 2005.

[2] H. Narayanan and S. Mitter, "Sample complexity of testing the manifold hypothesis," in *Advances in Neural Information Processing Systems 23*, J. Lafferty, C. Williams, J. Shawe-Taylor, and R. Z. a nd A. Culotta, Eds., Curran Associates, Inc., 2010, pp. 1786–1794.

[3] F. Palmieri and U. Fiore, "A nonlinear, recurrence-based approach to traffic classification," *Computer Networks*, vol. 53, no. 6, pp. 761–773, 2009.

[4] S. Ozair and Y. Bengio, "Deep directed generative autoencoders," Université de Montréal, Tech. Rep., 2014, http://arxiv.org/pdf/1410.0630.pdf.

[5] Y. Bengio, G. Mesnil, Y. Dauphin, and S. Rifai, "Better mixing via deep representations," in *Proceedings of the 30th International Conference on Machine Learning (ICML13)*, ACM, 2013, pp. 552–560.

[6] X. Glorot, A. Bordes, and Y. Bengio, "Domain adaptation for large-scale sentiment classification: a deep learning approach," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 513–520.

[7] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, ACM, 2008, pp. 1096–1103.

[8] U. Fiore, F. Palmieri, A. Castiglione, and A. De Santis, "Network anomaly detection with the Restricted Boltzmann Machine," *Neurocomputing*, vol. 122, pp. 13–23, 2013.

[9] G. E. Hinton, "Learning multiple layers of representation," *Trends in cognitive sciences*, vol. 11, no. 10, pp. 428–434, 2007.

[10] ——, "Training products of experts by minimizing contrastive divergence," *Neural computation*, vol. 14, no. 8, pp. 1771–1800, 2002.

[11] R. Salakhutdinov and G. Hinton, "An efficient learning procedure for deep boltzmann machines," *Neural computation*, vol. 24, no. 8, pp. 1967–2006, 2012.

[12] J. Goldberger, G. E. Hinton, S. T. Roweis, and R. Salakhutdinov, "Neighbourhood components analysis," in *Advances in Neural Information Processing Systems*, 2004, pp. 513–520.

[13] R. Salakhutdinov and G. E. Hinton, "Learning a nonlinear embedding by preserving class neighbourhood structure," in *International Conference on Artificial Intelligence and Statistics*, 2007, pp. 412–419.

Chapter 4

Independent Component Analysis

Finding a good representation of observed data, essentially, amounts to transform it until structure becomes more visible. A representation is better than another as far as it is able to capture more information about regularities and dependencies in the input data. In many cases of practical interest, and network anomaly detection is no exception, in the light of the complexity of the observed behavior, it can be expected that the underlying dynamics governing the process producing it are intricate as well. The regularities and dependencies shown on observed data can, thus, involve many input dimensions or combinations thereof.

Independent Component Analysis [1] aims at separating observed data into statistically independent components. ICA bears a close relationship to the problem of Blind Source Separation (BSS) in signal processing. BSS assumes that the observed data is the product of a mixing process acting on unknown components on which weak assumptions are made, and has the objective of finding those components. The observed signal can be reconstructed based on the components found. ICA is one method for addressing and solving the BSS problem. When some mixtures of probabilistically independent source signals are observed, ICA recovers the original source signals from the observed mixtures without knowing how the sources are mixed. A requirement for the signal reconstruction system is that it is should adaptively conform to nonstationary variations of the source and observed signals.

The way the original source signals can be extracted from the observed signals is, in general, not unique. For example, scaling of the original sources produces

indeterminacies that are difficult to avoid. Additionally, the order in which the source signals are found may not be determined. These indeterminacies are inherent to the fact that both the factors and the mixing function are unknown. The most relevant information, however, is in the waveform of the reconstructed signals rather than in their amplitude or order of arrangement in the reconstruction.

Let $\mathbf{x}(t) = [x_1(t), \ldots, x_n(t)]^T$ be a vector of n observations at index t. What is ideally sought is a mapping from the n-dimensional input space to an m-dimensional representation space such that the transformed variables are as close as possible to the factors that explain the structure in the observed data. Suppose that the observed variables are the result of a combination of (unknown) explaining factors, also called *sources*, $\mathbf{s}(t) = [s_1(t), \ldots, s_m(t)]^T$, with typically $m \leq n$:

$$\mathbf{x}(t) = \mathbf{f}(\mathbf{s}(t)) \tag{4.1}$$

through a mapping $\mathbf{f}$. The objective is to find a mapping $\mathbf{g}$ such that

$$\hat{\mathbf{s}}(t) = \mathbf{g}(\mathbf{x}(t)) \tag{4.2}$$

will be an estimate of the source vector $\mathbf{s}(t)$. The sources $\mathbf{s}(t)$ are assumed to be zero-mean and mutually independent:

$$\mathbf{s}_i(t) \perp \mathbf{s}_j(t) \quad \forall i \neq j \tag{4.3}$$

with $\perp$ denoting statistical independence. Note that any non zero-mean source can be modeled by the sum of a zero-mean source and an additive constant source.

The most frequently used data model is linear:

$$\mathbf{x}(t) = \mathbf{A}\mathbf{s}(t) \tag{4.4}$$

where $\mathbf{A}$ is an unknown matrix called *mixing matrix*. Consequently, the observed signals are also called mixed signals. In the linear model, the objective is to find a matrix $\mathbf{W}$ (the *separating matrix*) such that the linear transformation $\mathbf{y} = \mathbf{W}\mathbf{x}$ produces a reconstruction signal $\mathbf{y}(t) = [y_1(t), \ldots, y_n(t)]^T$ whose components are as

independent as possible. The weights, i.e. the elements w_{ij} of the separating matrix $\mathbf{W}$ need to be adapted in order to generate estimates of the source signals:

$$\hat{\mathbf{s}}(t) = \mathbf{y}(t) = \mathbf{W}\mathbf{x}(t)\,. \tag{4.5}$$

The matrix $\mathbf{W}$ that is estimated together with the sources is thus the pseudo-inverse of $\mathbf{A}$, that is, $\mathbf{W} = \mathbf{A}^{+}$.

4.1 Principal Component Analysis

A classical method to find an unmixing matrix is to determine the linear combination that explains the maximum variation in the observations. Put differently, find a representation of the data in a rotated coordinate system (the *principal components*) such that each axis is aligned (if data is zero-meaned) with the greatest variance not already accounted for by the preceding axes. Principal Component Analysis (PCA) was first introduced by Pearson [2] in the context of finding closest fits for sets of points in space when, in contrast to situations where the "independent" variables are supposed to be accurately known and the probable values of "dependent" variables are sought, all variables are found by experiment or observation.

The order in which principal components are arranged is significant: the first components are those that convey as much information as possible. One interpretation for this is that PCA can be used to separate signal from noise. The former will coincide with the directions on which input data have the largest variance, while the latter will be aligned with the directions accounting for the residual variance. In the same vein, another application of PCA has been dimensionality reduction. Again, the first principal components are good candidates to select a subset of features which are the most informative ones. The components with smaller variances can, instead, be discarded without losing too much information.

The use of PCA in anomaly detection of network traffic was proposed in [3] and extended in [4]. Those were also the first studies that focused on network-wide traffic

instead on concentrating upon single-link traffic. In [3], Lakhina *et al.* decomposed traffic volume on each backbone link as the linear sum of Origin-Destination (OD) flows traversing that link. Subsequently, by applying PCA on the traffic link matrix, they achieved to separate traffic into principal components that, when compared to baseline measurements, showed anomalies. The subsequent work [4] adopted an unsupervised approach and widened the analysis to encompass the entropies of origin and destination addresses and ports.

PCA can be converted to the eigen structure problem of the data covariance matrix[1] $\mathbf{R_{xx}} = E\{\mathbf{x}\mathbf{x}^T\}$, where $E\{\cdot\}$ denotes the expectation operator. Thus, PCA can be expressed as an eigenvalue problem [5]

$$\mathbf{R_{xx}}\mathbf{v}_j = \lambda_j \mathbf{v}_j \tag{4.6}$$

where λ_j are the eigenvalues of the covariance matrix, and $\mathbf{v}_j$ are the corresponding eigenvectors. Denoting by $\mathbf{\Lambda_x}$ the diagonal matrix of the eigenvalues and by $\mathbf{V_x}$ the orthogonal matrix of the eigenvectors, Eq. (4.6) can be written in matrix form

$$\mathbf{R_{xx}}\mathbf{V_x} = \mathbf{V_x}\mathbf{\Lambda_x} \tag{4.7}$$

or, since $\mathbf{V_x}$ is orthogonal,

$$\mathbf{V_x}^T\mathbf{R_{xx}}\mathbf{V_x} = \mathbf{\Lambda_x} \tag{4.8}$$

The covariance matrix is not available in practice, since the distribution of observed data is not known. What is available is an estimate $\hat{\mathbf{R}}_{\mathbf{xx}}$ of $\mathbf{R_{xx}}$ based on the N observed samples

$$\hat{\mathbf{R}}_{\mathbf{xx}} = \frac{1}{N}\sum_{t=1}^{N}\mathbf{x}(t)\mathbf{x}^T(t) \tag{4.9}$$

It is essential for PCA that there is redundancy in the observations. Had the analyzed components been independent, PCA would have been useless.

[1]The covariance matrix is the correlation matrix with the mean removed; for zero-mean signals such as the one considered here, the covariance matrix and the correlation matrix are identical.

4.2 Centering and Whitening

Unlike PCA, all ICA methods rely on higher-order statistics in some form. Expressed in a different way, they are based on information not contained in the covariance matrix. Preprocessing is an important step in ICA, and basically amounts to removing the effects of first-order and second-order statistics.

It is useful to consider zero-mean random vectors, because for such vectors, the covariance and correlation matrices are the same. In practice, *centering* consists in subtracting the sample mean from observations:

$$\mathbf{x}' = \mathbf{x} - \bar{\mathbf{x}} \tag{4.10}$$

so that $E\{\mathbf{x}'\} = \mathbf{0}$.

Some algorithms require *prewhitening* of mixed signals. A random, zero-mean n-vector $\mathbf{u}$ is said to be *white* if its covariance matrix is the $n \times n$ identity matrix $\mathbf{I}_n$:

$$\mathbf{R}_{\mathbf{uu}} = E\{\mathbf{u}\mathbf{u}^T\} = \mathbf{I}_n \,. \tag{4.11}$$

Thus, the components of a white vector are mutually uncorrelated and have unit variance: $E\{u_i u_j\} = \delta_{ij}$ by the definition of $\mathbf{R}_{\mathbf{uu}}$ and Eq. (4.11), where δ_{ij} is the Kronecker delta. A whitening transformation is a linear transformation of a random vector $\mathbf{x}$ such that the resulting vector is white. If the whitening transformation is written as

$$\mathbf{u}(t) = \mathbf{Y}\mathbf{x}(t) \tag{4.12}$$

then the whitening matrix (also known as the Mahalanobis decorrelation transformation [5]) $\mathbf{Y}$ should be chosen in such a way that the covariance matrix $E\{\mathbf{u}\mathbf{u}^T\}$ is the identity matrix.

Whitening fundamentally consists in decorrelation followed by scaling, and PCA gives a solution. Indeed, the matrix

$$\mathbf{Y} = \mathbf{\Lambda_x}^{\frac{1}{2}} \mathbf{V_x}^T \tag{4.13}$$

is a whitening matrix. Recalling Eq. (4.8) and Eq. (4.13)

$$\begin{aligned}\mathbf{R_{uu}} &= E\{\mathbf{u}\mathbf{u}^T\} = E\{\mathbf{Y}\mathbf{x}\mathbf{x}^T\mathbf{Y}^T\} = \mathbf{Y}\mathbf{R_{xx}}\mathbf{Y}^T = \mathbf{Y}\mathbf{V_x}\mathbf{\Lambda_x}{\mathbf{V_x}}^T\mathbf{Y}^T \\ &= {\mathbf{\Lambda_x}}^{-\frac{1}{2}}{\mathbf{V_x}}^T\mathbf{V_x}\mathbf{\Lambda_x}{\mathbf{V_x}}^T\mathbf{V_x}{\mathbf{\Lambda_x}}^{-\frac{1}{2}} = {\mathbf{\Lambda_x}}^{-\frac{1}{2}}\mathbf{\Lambda_x}{\mathbf{\Lambda_x}}^{-\frac{1}{2}} = \\ &= {\mathbf{\Lambda_x}}^{-\frac{1}{2}}{\mathbf{\Lambda_x}}^{\frac{1}{2}}{\mathbf{\Lambda_x}}^{\frac{1}{2}}{\mathbf{\Lambda_x}}^{-\frac{1}{2}} = \mathbf{I} \end{aligned} \tag{4.14}$$

because $\mathbf{V_x}$ is orthogonal and $\mathbf{\Lambda_x}$ is diagonal.

4.3 Orthogonalization

PCA or linear ICA will produce a set of source vectors such that all observed vectors can be expressed as a linear combination of the sources. Therefore, the sources will form a basis for the space spanned by the observations. As with any basis, a desirable property for the basis vectors produced by separation algorithms is that they are orthogonal or orthonormal. Not every algorithm will yield orthogonal vectors automatically. Therefore, an orthogonalization procedure should be applied to ensure that the solution obtained is an orthogonal basis for the space spanned by the observed data. Orthogonalization transforms a generic basis (i.e., a set of vectors such that any vector in the spanned space can be expressed as a linear combination of basis vectors) into an orthogonal basis. Orthonormalization is, then, achieved by normalizing each basis vector, i.e., dividing it by its Euclidean norm.

There are two main approaches to orthogonalization in ICA: deflationary and symmetric. *Deflationary* approaches are sequential in nature. Given an arbitrary basis $\{\mathbf{u}_1, \ldots, \mathbf{u}_n\}$, the classical Gram-Schmidt procedure iteratively takes at each step an orthogonal basis $\{\mathbf{v}_1, \ldots, \mathbf{v}_j\}$, $j < n$, for a subspace of the space spanned by the original basis and augments it, covering the entire space in the end. The algorithm starts with $\mathbf{v}_1 = \mathbf{u}_1$ and at each iteration step it adds to the set of orthogonal vectors $\{\mathbf{v}_1, \ldots, \mathbf{v}_{i-1}\}$ the vector $\mathbf{v}_i$:

$$\mathbf{v}_i = \mathbf{u}_i - \sum_{k=1}^{i-1} \frac{\mathbf{v}_k^T\mathbf{u}_i}{\|\mathbf{v}_k\|}\mathbf{v}_k \tag{4.15}$$

where $\|\mathbf{v}\| = \mathbf{v}^T\mathbf{v}$ denotes the norm. Orthogonality is maintained because, when checking $\mathbf{v}_j^T\mathbf{v}_i$ for any $j < i$, all the inner products in the sum vanish except the one for $k = j$ (as a consequence of the orthogonality of $\{\mathbf{v}_1, \ldots, \mathbf{v}_{i-1}\}$), and for that one the result is $\mathbf{v}_j^T\mathbf{u}_i - \mathbf{v}_j^T\mathbf{u}_i = 0$. Inherent to the sequential nature of the procedure are the problems of the proper choice of the order in which basis vectors are to be processed and the problem of accumulation of errors.

In contrast to deflation approaches, *symmetric* orthogonalization methods deal with all input basis vectors equally. Symmetric methods rely on the eigenvalue decomposition of the matrix obtained by stacking the basis vectors $\mathbf{u}_i$ columnwise.

4.4 Independence

Alternatively to uncorrelatedness, the principle of statistical *independence* can be used to disentangle the explanatory factors in observed data. If the sought factors reflect causes that were involved in the process that generated the observed data, it is reasonable to expect that a good representation would isolate a minimal number of independent factors. Being independent is a stronger condition than being uncorrelated: independence means that no information can be gained on the value of a factor, given the value of another factor. Independence can be defined referring to probabilities. Two random variables are independent if

$$\Pr(X = x, Y = y) = P(x, y) = P(x)P(y) \tag{4.16}$$

that is, the joint probability factorizes into the product of the marginal probabilities. Considering the conditional probabilities,

$$P(x|y) = \frac{P(x, y)}{P(y)}, \quad P(y|x) = \frac{P(x, y)}{P(x)} \tag{4.17}$$

it follows from Eq. (4.17) that

$$P(x|y) = P(x), \quad P(y|x) = P(y) \tag{4.18}$$

if x and y are independent.

For two independent random variables x and y, the following holds

$$E\{g(x)h(y)\} = E\{g(x)\}E\{h(y)\} \tag{4.19}$$

for any functions $g(\cdot)$ and $h(\cdot)$ such that the expectations exist. In other words, nonlinear transformations of independent function are uncorrelated. From Eq. (4.19) it can be seen that independence implies uncorrelatedness. The converse is not true, except in the case of variables having Gaussian distributions.

One of the methods to extract the independent components from a mixture is to find a separating matrix such that different components y_i and y_j are uncorrelated and, for suitable nonlinear functions g and h, $g(y_i)$ and $h(y_j)$ are uncorrelated. The other main method relies on the fact that the Central Limit Theorem states that the distribution of the sum of independent random variables tends to be gaussian, under mild conditions. Thus, to estimate the independent components, one can consider a linear combination of the input vectors and vary the coefficients until the maximum nongaussianity is obtained. The strategy of maximizing the nongaussianity can also be interpreted as minimizing the mutual information between the latent variables. The linear combination of independent sources is expected to be more gaussian than the sources, thus the maximum nongaussianity indicates that the linear combination is roughly aligned with one of the sources. The local maxima of nongaussianity correspond to independent components.

To obtain a quantitative measure of the nongaussianity of a random variable, a property of gaussian variables can be used, namely that they have the largest entropy among all variables with zero mean and unit variance. This result can be generalized: the gaussian distribution has the uniquely largest entropy among all distribution with a given covariance matrix. Entropy can therefore be used as a measure of nongaussianity. A convenient form of this measure of nongaussianity is the *negentropy*. The negentropy of a random vector $\mathbf{x}$ is defined as the difference between the entropy $H(\mathbf{x}_{gauss})$ of a gaussian random vector $\mathbf{x}_{gauss}$ having the same covariance as $\mathbf{x}$ and the entropy of $\mathbf{x}$:

$$J(\mathbf{x}) = H(\mathbf{x}_{gauss}) - H(\mathbf{x})\,. \tag{4.20}$$

The negentropy $J(\mathbf{x})$ is always nonnegative (due to the maximality of the entropy of the gaussian distribution) and is zero only if $\mathbf{x}$ is gaussian.

Estimating the negentropy is computationally very difficult. A computationally simple approximation of the negentropy of a scalar zero-mean random variable x is given by

$$J(x) \approx \frac{1}{12} E\{x^3\}^2 + \frac{1}{48} \text{kurt}(x)^2 \tag{4.21}$$

where $\text{kurt}(x) = E\{x^4\} - 3(E\{x^2\})^2$ is the kurtosis (x is zero-mean). The kurtosis measures the flatness of a distribution. Positive values of the kurtosis are associated to supergaussian (or leptokurtic) probability density functions with spiky peaks and heavy tails; negative values to subgaussian or platykurtic distributions with broad peaks and slender tails; zero kurtosis characterizes distributions peaked in the same way as a Gaussian distribution. Note that for white data, $\text{kurt}(x) = E\{x^4\} - 3$. The kurtosis itself could have been used as a measure on nongaussianity. It is, however, not robust with respect to outliers. Negentropy is less sensitive to irrelevant or erroneous data. Approximations to negentropy through the kurtosis attempt at combining the best characteristics of both indicators.

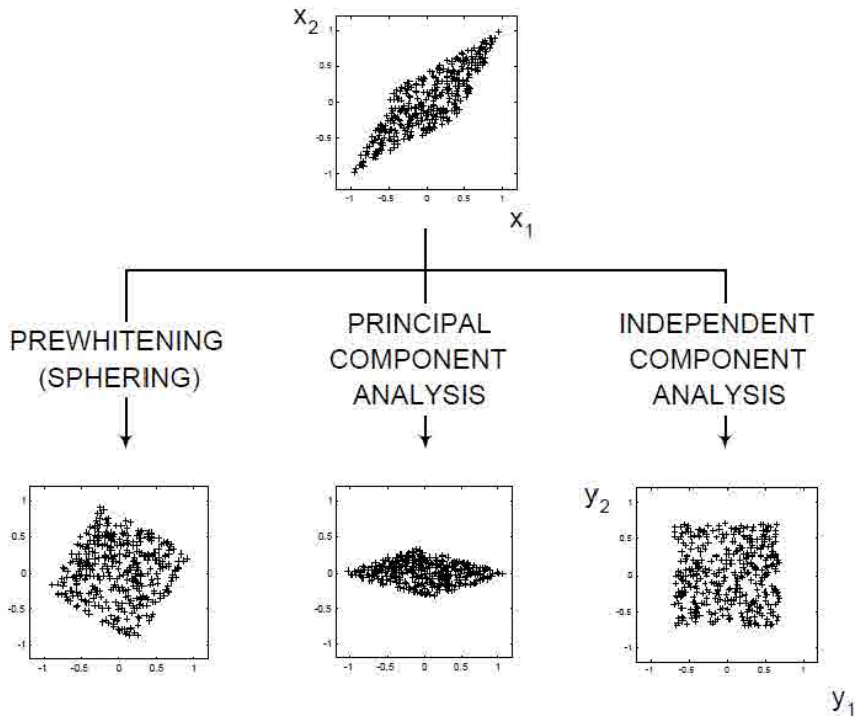

Figure 4.1: Illustration of the difference between transformations for a uniformly distributed bidimensional signal [5]

References

[1] A. Hyvärinen and E. Oja, "Independent component analysis: algorithms and applications," *Neural networks*, vol. 13, no. 4, pp. 411–430, 2000.

[2] K. Pearson, "On lines and planes of closest fit to systems of points in space," *Philosophical Magazine*, vol. 2, pp. 559–572, 6 1901.

[3] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," in *ACM SIGCOMM Computer Communication Review*, ACM, vol. 34, 2004, pp. 219–230.

[4] ——, "Mining anomalies using traffic feature distributions," in *ACM SIGCOMM Computer Communication Review*, ACM, vol. 35, 2005, pp. 217–228.

[5] A. Cichocki and S.-i. Amari, *Adaptive blind signal and image processing: learning algorithms and applications*. John Wiley & Sons, 2002.

Chapter 5

Improving anomaly detection

The shortcomings of signature-based systems become evident when such systems are confronted with previously unseen attacks, which can be completely new, variations of earlier attacks, or polymorphic ones. For an appropriate signature to be generated, the community needs to be made aware of the threat, and detailed information about it and its workings should become available. This time-consuming process does not make signatures a colorable alternative when prompt response is required. On the other side, network anomaly detection systems have the essential benefit of not requiring prior knowledge about attacks. However, they suffer from the absence of an universal traffic model which is valid anywhere and anytime. In addition, there is an inherent difficulty due to the fact that, while individual flows are often provoking or indicating an anomaly, only aggregate quantitative measures show the regularity needed for building a model for normal behavior and detecting variations from that model. The benefits of signature-based methods can be reaped by arranging multiple detectors in a hierarchical structure, where the front layers have the purpose of reducing the volume of traffic by quickly identifying the easily detectable attacks, easing the job of subsequent layers, which can perform more sophisticated analyses on a subset of flows.

Whatever the strategy that is selected to detect attacks, traffic should be captured and observed at some place in the network. An important point that should be scrutinized is the choice of how many sensing points to deploy and where those should be placed for the performance of the anomaly detection system to be opti-

mal. Interfaces in the routers at the network border are an appealing option, as they allow the capture of traffic traversing the border. Such choice would, however, prevent the observation of activity which is only "internal" to the network considered, and threats coming from the inside can outrank their outside counterparts both in number and in harmfulness [1]. In addition, if the collection points are too close to the backbone, it is possible that excessive aggregation of traffic information results in losses in discriminative power. A single collection point can only build a partial view of traffic. Ideally, many different sensing points are needed, at different hierarchical levels, to obtain a rich and complete set of measurements able to describe traffic in a comprehensive way.

Accordingly, we propose a novel network anomaly detection architecture, relying on a fully distributed traffic observation system based on multiple independent sensor agents installed on a large number of connected hosts. These sensors collect traffic reaching their network interfaces, pre-process it by extracting some features of interest that should best describe the involved traffic dynamics, and finally send the resulting data to one (or two, for redundancy/robustness sake) centralized collector node aggregating all the data incoming from the distributed sensors for further analysis.

5.1 Different views on the same phenomenon

Different features may reflect different observation dimensions such as the communication intensity (i.e., number of transactions/flows, average packet length, packet inter-arrival times, etc.), and connection dispersion (i.e., number of corresponding counterparts). By using these features, each sensor/host is able to describe in a detailed and independent way the traffic-related phenomena observable from its own point of view. The resulting multi-dimensional data exhibit distributions that are highly sparse and may lack of invariance in their statistical distributions with respect to the associated generating events/phenomena. That is, the sequence of samples

associated to the same kind of events occurring at different times may be mapped in completely different classes of traffic behavior. Furthermore, the different samples/signals describing the observed features may have a disordered structure and tend to be highly redundant, presenting overlapping and mutual dependencies.

These signals may be depending on other, more intrinsic, variables determined by the hidden dynamics governing the traffic itself so that each individual observation may be viewed as a linear combination of the above variables. Such variables can be reasonably assumed to be mutually independent and their number is a fundamental dimension to be used to correctly describe traffic, and hence to flag anomalous behaviors. That is, each intrinsic component, due to the above independence property, contains a great deal of information about the real traffic dynamics, and hence, being strictly necessary to describe it, can be more sensitive to network anomalies and their related disorders than other directly available features. Such more sensitive components, incoming from the different sensors, can be considered as the really useful signals or "fundamental" features to be inspected in the classification of anomalous or normal phenomena, while the components associated to the other correlated or non-informative features can be viewed as "noise" or "unwanted" signals, degrading the classifier's performance.

Hence, we are interested in extracting the network behavior profiles to be used for a more effective classification process from the time series associated to the above intrinsic components, determined by performing ICA on the observation data collected from all the available sensors. This can be modeled as a BSS problem, such as the classic n by m "cocktail party" one, where the n individual speaker (or "voice") signals to be isolated are the fundamental traffic components and the m microphones, capturing the mixed voices and noise, correspond to the available traffic sensors. The power of this approach resides in the assumption that different processes independently operating on the network (e.g., browsing, data transfer, e-mail retrieval or peer-to-peer sessions) generate unrelated signals, whose statistical combination completely describes the network behavior and hence the desired "baseline". In other words, the observed traffic results from the superposition of a certain number

of fundamental components, directly associated to these unrelated signals. Due to their mutual independence the resulting component signals inherently maximize the information content that can be used to describe the traffic behavior and its less evident dynamics, and hence, their exploitation is expected to introduce significant improvements in the overall efficiency of the anomaly classification process without loss of accuracy.

Starting from these "fundamental" signals/features, we can try to "understand" the process generating the traffic data, and hence complete our baseline modeling task, by allowing the system to build its knowledge about the occurrence of the statistical phenomena describing the "normal" traffic via inductive inference based on observing the empirical data that represent incomplete information about their evolution dynamics. This will be the training activity in which the system "learns" by example to recognize complex patterns, to distinguish between samples based on their different patterns and to make intelligent decisions related to binary classification of the anomalous and normal events. The training data consist of a set of pre-classified independent component signals (or time series), corresponding to the above fundamental features. According to a classic supervised learning paradigm, each training instance is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal). A supervised learning algorithm analyzes such training data and produces an inferred function, which drives the classifier (if the output is discrete). The inferred function should predict the correct output value for any valid input object. This requires the learning algorithm to generalize from the known pre-classified situations in training data to unseen situations in a "reasonable" way.

The learning algorithm may be structured according to a decision tree scheme, such as C4.5 and related methods, generating a sequence of rules based on splitting its input objects into the two classes *Normal, Anomalous* according to the above pre-trained model, that is able to work in a satisfactory way also on unseen input values (i.e., it generalizes well).

5.2 Baselining and Features Extraction

A fundamental preliminary task in a network anomaly detection process is baselining, which can be defined as the act of measuring and rating the performance of a network under normal conditions. Providing a network baseline requires evaluating the regular network utilization, protocol usage, peak network utilization, and average throughput over a significant period of time. This is a very slow and complex task requiring a lot of computing effort and human expertise, but fortunately it has to be performed only once, in the initial "knowledge construction" phase where the most significant network utilization patterns, describing the fundamental traffic dynamics, should be described in terms of specific features gathered from traffic observations.

In building our baseline, we start from the assumption that traffic originating from a single host results from the linear composition of many independent streams, each associated to a specific instance of a protocol/application (e.g., web browsing, mail activity, file sharing, etc.). The same concept can be extended to the aggregated traffic associated to a group of hosts or an entire network. In this case, each stream can be viewed as the composition of effects due to homogeneous protocol/application-originated traffic belonging to the involved hosts. The composition affects all the statistical features measured, though not every feature is affected in the same way. Starting from the features associated to cumulative traffic trends, the simultaneous analysis of the above independent streams related to the cumulative variations in each traffic class or host aggregates, introduces great control granularity in the detection process. The time series incoming from multiple points of observation, and hence multiple dimensions in the feature vectors, can ease correlation and inference activities in the machine learning based binary classification process. The availability of several different observations (coming from multiple sensors) associated to the individual traffic features may also be helpful in spotting and describing the nature and behavior of the observed anomalous phenomena. For example, the analysis of an anomaly can be, in principle, extended

to encompass the protocols that are involved, the transport facilities that are used, and the distribution of traffic volumes distribution.

We can note that under normal network conditions the traffic time series associated to each aggregation of homogeneous streams tend to systematically follow a specific trend/distribution when observed on a sufficiently large network dimension and time scale. Accordingly, we transform the aggregated input data into a subset of features so that the reduced representation contains only the really relevant information from the original signal dynamics and no more artifacts or noise effects due to the combination of mutually dependent signals. For this reason, the resulting signals/time series will be the ones bringing the maximum content of information. Consequently, they will be the most relevant and effective, or "fundamental", features to be used in the classification process.

In doing this we used the CAIDA CoralReef tools [2] to process the collected packet traces and compute the original input feature values. CoralReef is a suite of tools developed by CAIDA to assist network administrators. The suite includes device drivers, libraries, and applications that support the capture and analysys of network traffic in a wide variety of formats and systems. The suite is designed to be easily expandable: by using the available APIs, new tools can be developed with limited effort.

The basic features considered in our scheme are any statistics (e.g., traffic volume data, the lengths of observed IP packets, the packet inter-arrival times, etc.) that can be useful to characterize the behavior of network traffic, structured as multi-dimensional feature vectors. When choosing such features, we only considered statistics that can be easily and effectively computed by the distributed sensors, without introducing additional burden—unsustainable on resource-constrained devices. This enables our system to be completely agnostic with regard to packet payload as well as to implicit constraints introduced from the transport layer. Accordingly, the statistics considered in constructing our basic feature set are:

- the average packet length within the time interval. Lengths are based on the IP datagram length excluding link layer overhead;

- the average packet arrival rate within the sampling interval;
- the average number of end-to-end transactions or traffic flows.

Several other more detailed features can be determined by breaking traffic down into elementary flows, which are specifically identified by source and destination IP addresses, protocol, source and destination ports. However, these data cannot be completely relied upon, and hence must be used carefully. Addresses can be spoofed, especially when one of the interacting parties has no interest in the other's reply, as it is the case with many DoS attacks. Proxies and NAT boxes can also introduce noise within a traffic description plan of action based solely on the address. Port numbers can be easily spoofed, too: if the communicating entities agree, they can select arbitrary port numbers, irrespective of the ones usually assigned to the application protocol used. Even tunneling can be used for the purpose of disassociating port numbers with application traffic. The traffic relative to one (tunneled) application can be encapsulated within traffic belonging to another (tunneling) application, thus maintaining the properties of the tunneling one. Masquerading techniques such as the above have been extensively used in order to make unauthorized, or undesirable, traffic appear as legitimate traffic. Abundant literature is devoted to the problem of characterizing traffic based on its properties and behavior, thus spotting port number disguise [3], [4]).

Starting from the basic features, we try to gather the characteristic properties of the most significant traffic components by extracting a minimum set of highly discriminating (or "pure") traffic features on which the various traffic profiles can be reliably built. More specifically, for each measured traffic feature, the corresponding time series values, sampled across all the available observation points (the sensors), should result from the linear composition of the "pure" (or fundamental) traffic features deriving from all the individual and independent component signals. Therefore, although the single components may not strictly adhere to a specific distribution, if we assume that in line of principle the component series are generated by independent, non-gaussian, stochastic processes, we can estimate, through ICA,

the single component signals that can be associated to the basic protocol/application streams (i.e., the "voices" to be separated in the cocktail-party problem), starting from the original multivariate "noisy" traffic measures. An example of such "voice" or basic independent component signal, may be the time series corresponding to all the web access-related traffic.

5.3 FastICA

In this work, the FastICA [5] method has been used to separate the independent components, where the non-gaussianity is measured by means of negentropy, defined in Eq. (4.20) and approximated as described in Eq. (4.21).

In contrast to many ICA techniques, based on other measures of non-gaussianity, the convergence of the FastICA algorithm is cubic (under mild conditions on the nonlinearity chosen) and the independent components can be easily estimated one by one. Convergence, with gradient-based methods, is slow and critically depends on the learning rate. In gradient descent methods, the learning rate gives the step taken in the opposite direction of the gradient of the objective function. If the learning rate is too low, convergence will be very slow. Conversely, a high learning rate may lead to instability. By contrast, FastICA does not require setting a learning rate nor it has adjustable parameters. The algorithm is asymptotically efficient if the non-linearity is optimally matched to the probability density function of the source [6]. Implementations of FastICA are available in a variety of programming languages, including C++ and Python. Both the deflation and symmetric approaches to orthonormalization are available, in literature and in software. The convergence properties, originally derived for the deflation version of the algorithm, are also shared by the symmetric version [7].

In our specific case, the traffic time series resulting from the available observations can be viewed as made up of a finite number of component signals $\mathbf{s}$—mixed through an unknown linear mixing process described by an $n \times n$ mixing matrix $\mathbf{A}$ as

in Eq. (4.4)—while the n different observed dimensions (the traffic features) record the mixed signals $\mathbf{x}$. Each of the components changes in time, but has a fixed weight for each channel. The FastICA algorithm finds an unmixing (separating) $n \times n$ matrix $\mathbf{W}$ consisting of the coefficients with which the traffic feature signals should be taken to form, by summation, the estimated components $\hat{\mathbf{s}}$ as in Eq. (4.5). In other terms, these coefficients indicate how strongly each involved observation brings information that is strictly required to describe the traffic dynamics.

5.4 Knowledge Construction: Rule-Based Classifier

Once we have been able to estimate the fundamental (i.e., unmixed) traffic components, the description of traffic profiles built from the time series of feature vectors, considered with reference to a consolidated baseline, can help us to isolate previously unseen, and hence possibly anomalous, behaviors. The construction of the knowledge base from the aforementioned feature vectors/profiles, determined according to the previously presented methods, is the next step in building a working anomaly detection system. To this purpose, a sufficiently large number of pre-classified feature vector samples have been aggregated into homogeneous "training" data sets. With the aid of traditional data mining and machine learning techniques, training data are used to determine the most discriminating features associated to the interesting type of traffic. Hence, a model of normal and anomalous traffic data is built from the training set, and the anomaly detector will take its decisions by inference from this model, being able to classify previously unseen events also without knowing what they look like.

More formally, training data is a set $T = \{\mathbf{t}_1, \mathbf{t}_2, \ldots \mathbf{t}_N\}$ of N already classified samples. Each sample $\mathbf{t}_i = (t_1^{(i)}, t_2^{(i)}, \ldots t_n^{(i)})$ is an n-dimensional vector where the j-th component $t_j^{(i)}$ represents the j-th independent feature, extracted by ICA, from the original sample observations incoming from the input sensors. The training data

is augmented with an additional feature set $C = (c_1, c_2, \ldots c_T)$ where c_i represent the class (i.e., anomalous/not anomalous) to which sample $\mathbf{t}_i$ belongs. In this way, the detector's knowledge could be modeled as a set or rules/criteria "*learned*" from a collection of training samples gathered under "*certified*" normal or anomalous conditions.

Decision trees are currently among the best available solutions to implement inductive rule-based classification schemes, where the decision process is expressed as a recursive partition of the instance space [8]. The widespread use of decision trees is essentially due to their robustness and execution performance, as well as to their ability in translating concepts into explicit rules that can be interpreted and checked with modest effort. Decision trees are structured according to the choice steps needed in order to achieve a correct classification. Nodes within the tree correspond to a test. These tests can involve the weighing of a single specific feature against a constant, the comparison of two features with each other, or the evaluation of a function of a collection of features. Branches correspond to possible outcomes of the test. Each decision step results in a move down the tree. The path taken describes the decision process needed to complete the classification task for a specific instance. Leaf nodes provide the class labels that are predicted for paths that arrive at the leaf. Put differently, classification using a decision tree always starts with a root node, considered to be the "ancestor" of every other decision node, and is accomplished recursively, by following the branches corresponding to the values that can be assumed by the features involved into the decision process. Hence, starting from the root node, one arrives at a leaf whose class value is assigned to the instance to be classified. Each path from the root of a decision tree to one of its leaves can be transformed into a rule: the antecedent part is formed by joining the comparisons along the path and the class prediction in the leaf is taken as the classifier output.

In order to create the most efficient decision tree, the simplest method uses a greedy approach, driven by a divide-and-conquer strategy. Starting with an empty tree, the algorithm greedily selects a first attribute to partition the training instances

into subsets associated to the possible attribute values. It then creates a node (initially, the root) corresponding to splitting the training set on the selected attribute. Afterwards, child nodes are created recursively by processing the split sets, stopping when the sets are pure (i.e., all instances belong to the same class) or there are no more features to be scrutinized.

It is likely that the greedy strategy for constructing decision trees will produce trees that mirror too closely the detailed structure of training data. Artifacts of the particular distribution of the training data used to build the decision tree may blend with characteristics genuinely related to the general distribution of the data of interest, interfering with the ability to generalize to unseen test data. In a word, overfitting is a possible problem with trees built by the divide-and-conquer algorithm. Although this may be somewhat alleviated by n-fold cross-validation, strategies have been developed for pruning decision trees [8]. These methods evaluate the effects of two basic pruning operations, namely replacing a subtree with a single leaf and collapsing a path by removing a branch and raising the subtree below it (see Fig. 5.1). Both operations will induce an increase in the classification errors on the training data, but they may lead to improving the performance on an arbitrary test sets. The criterion to guide pruning decisions needs an estimate of the error that can be expected at an internal node on generalized test data. If the estimated error for a candidate replacement subtree is lower than that of the original subtree, then replacement is advisable.

We used one of the most common classification schemes based on decision trees, formalized within the well-known C4.5 algorithm [9] whose pseudo-code formulation is reported in Algorithm 1. At each node of the tree, the C4.5 algorithm determines, by the locally most effective choice, the attribute that splits in the best way the involved instances into different subsets associated to the different classes. The information that is gained by splitting on an attribute is a measure of the effectiveness of splitting on that attribute. Thus, an appealing selecting criterion is the Information Gain (IG)—difference in entropy (see also Section 6.1)—that results from choosing an attribute for splitting the data. The IG should be corrected, however, to

Algorithm 1 C4.5Tree(R) – Building a C4.5 tree

Input:

R: current decision tree root

Output:

N: resulting decision tree root

```
1: Check for base cases
2: N ← new decision node
3: maxig ← 0
4: abest ← nil
5: for all attribute a do
6:    Find the Gain Ratio IGR(a) from splitting on a
7:    if IGR(a) ≥ maxig then
8:       maxig ← IGR(a)
9:       abest ← a
10:   end if
11: end for
12: Create a decision node that splits on abest
13: for all U in the splitting of R do
14:   if U is not empty then
15:      add C4.5Tree(U) as a child of N
16:   end if
17: end for
18: return N
```

account for the effects deriving from the nonuniformity in the number of partitions induced by attributes. The information gain attainable by splitting on attributes with a large number of possible values tends, in fact, to outrank the one obtainable by splitting on attributes with less options. The gain ratio (the IG divided by the entropy of the split obtained without considering class labels) is therefore chosen as the quantity to maximize when making the selection of the attribute on which to split. The C4.5 algorithm then recurs on the smaller subsets produced, terminating when a given subset is pure. The resulting decision tree contains all the features that are most predictive to classification, whereas the branches that correspond to less useful features are automatically pruned (see Fig. 5.1). Hence, irrelevant and redundant information is dealt with by providing an automatic feature selection strategy based on an initial learning/mining phase operating on a sufficiently large quantity of pre-classified data. It also provides the basic mechanisms for ranking (usually on the basis of the Information Gain) the features present in the decision tree to determine the relative importance of any individual feature.

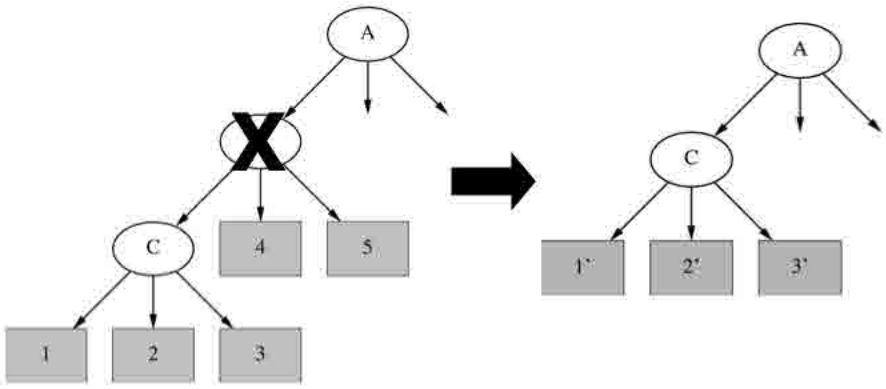

Figure 5.1: Pruning on the decision tree example (source: [8]).

The aforementioned IG comes from information theory, and represents the change in entropy from a prior state to a state that takes some information as given. Denoting with $T_v^{(a)} \subseteq T$ the subset of the training set such that the value of attribute

a is v, the IG is

$$IG(X,a) = H(T) - \sum_{v \in D_a} \frac{|T_v^{(a)}|}{|T|} H(T_v^{(a)}) \tag{5.1}$$

where D_a is the domain of a and H is the entropy. The IG is thus associated to an increment in certainty resulting from a splitting decision. Ideally, each split in the decision tree should drive the classification to the most efficient choice, and that choice is the one that reduces randomness by the greatest amount. The whole process of training and building a decision tree-based binary classifier is sketched in Algorithm 2.

Algorithm 2 *Training* $\left(t, n, m, \left\{\{\vec{x}_j^1\}_{j=1}^t, \ldots, \{\vec{x}_j^m\}_{j=1}^t\right\}, p\right)$ – The training process

Input:

t: training duration in time samples

n: number of independent components to be isolated, $n \geq m$

m: number of sensors

$\left\{\{\vec{x}_j^1\}_{j=1}^t, \ldots, \{\vec{x}_j^m\}_{j=1}^t\right\}$*: basic feature vectors, where* $\vec{x} = \{f_1, \ldots, f_p\}$

p: number of basic features in vectors $\vec{x}$

Output:

DT: C4.5 decision tree classifier

Local:

$\left\{\{y(f)_j^1\}_{j=1}^t, \ldots, \{y(f)_j^n\}_{j=1}^t\right\}$*: independent components resulting from ICA on feature f*

$\left\{\{\vec{z}_j^1\}_{j=1}^t, \ldots, \{\vec{z}_j^n\}_{j=1}^t\right\}$*: fundamental feature vectors*

1: **if** $n > m$ **then**

2: $n \leftarrow m$ {ICA requires the components being less than sources}

3: **end if**

4: $\{\{\vec{z}^1\}, \ldots, \{\vec{z}^n\}\} \leftarrow \{\varnothing, \ldots, \varnothing\}$

5: **for all** *feature* $f \in \vec{x}$ **do**

6: $\left\{\{y(f)_j^1\}_{j=1}^t, \ldots, \{y(f)_j^n\}_{j=1}^t\right\} \leftarrow ICA\left(t, m, n, \left\{\{f_j^1\}_{j=1}^t, \ldots, \{f_j^m\}_{j=1}^t\right\}\right)$ {extract components}

7: $\left\{\{\vec{z}_j^1\}_{j=1}^t, \ldots, \{\vec{z}_j^n\}_{j=1}^t\right\} \leftarrow \left\{\{\vec{z}_j^1 \cup y(f)_j^1\}_{j=1}^t, \ldots, \{\vec{z}_j^n \cup y(f)_j^n\}_{j=1}^t\right\}$ {build fundamental feature vectors}

8: **end for**

9: $DT \leftarrow BuildDecisionTree\left(t, n, \left\{\{\vec{z}_j^1\}_{j=1}^t, \ldots, \{\vec{z}_j^n\}_{j=1}^t\right\}\right)$

10: **return** *DT*

References

[1] E. E. Schultz, "A framework for understanding and predicting insider attacks," *Computers & Security*, vol. 21, no. 6, pp. 526–531, 2002.

[2] K. Keys, D. Moore, R. Koga, E. Lagache, M. Tesch, and k. c. Claffy, "The architecture of CoralReef: an internet traffic monitoring software suite," in *Passive and Active Network Measurement Workshop (PAM)*, Amsterdam, Netherlands: RIPE NCC, 2001.

[3] Y. Wang, Y. Xiang, and S. Yu, "An automatic application signature construction system for unknown traffic," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 13, pp. 1927–1944, 2010.

[4] F. Palmieri and U. Fiore, "A nonlinear, recurrence-based approach to traffic classification," *Computer Networks*, vol. 53, no. 6, pp. 761–773, 2009.

[5] A. Hyvärinen and E. Oja, "A fast fixed-point algorithm for independent component analysis," *Neural computation*, vol. 9, no. 7, pp. 1483–1492, 1997.

[6] P. Tichavsky, Z. Koldovsky, and E. Oja, "Performance analysis of the fastica algorithm and crame´ r-rao bounds for linear independent component analysis," *Signal Processing, IEEE Transactions on*, vol. 54, no. 4, pp. 1189–1203, 2006.

[7] E. Oja and Z. Yuan, "The fastica algorithm revisited: convergence analysis," *Neural Networks, IEEE Transactions on*, vol. 17, no. 6, pp. 1370–1381, 2006.

[8] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[9] J. Quinlan, *C4.5: programs for machine learning*. Morgan Kaufmann, 1993, vol. 1.

Chapter 6

Experimental evaluation

As it has been discussed in the previous chapters, the basic intuition behind the use of ICA to untangle the explanatory factors of traffic and achieve a better classification performance in anomaly detection is two-sided. The first part is that the data we observe are the result of a complex process, where several factors play a role in an intertwined pattern of interactions. In particular, observed quantities at the aggregate level are thus the superposition of the effects of simpler streams. The second part is the hypothesis that extraction of independent components from these aggregated measurements is a step towards the construction of a simplified model. The view of independent components thus unveils, somehow, portions of information which were inaccessible because they were masked by the intricate webwork of relationships between the explanatory factors.

To test this hypothesis, the performance of some anomaly detectors has been analyzed, comparing the results obtained in two situations. In the first setting, the classifiers were given the original, unprocessed data. In the second setting, the independent components extracted from the original data were given as input to the classifiers. An improvement in the performance of classifiers would mean that some information had been made available by the application of ICA in such a way that anomalies that had previously gone undetected can now be discovered. In particular, especially a reduction in false negatives would be an indicator that the model that can be built after ICA is closer to the true data distribution. In other words, the manifold where normal data lie has, after ICA, a simpler structure

allowing a more accurate separation between points belonging to the manifold and points not belonging to it.

In order to verify the effectiveness of the models induced by our changes of representation in improving anomaly detection, we carried out an extensive set of experiments on real network traffic traces (see Table 6.1) collected at the University of Naples, Italy. A proof-of-concept implementation has been built, which did not require the use of proprietary analysis tools: rather, it has been based on open-source software, because of the wide availability of these tools and of the rich set of functionalities they provide. In our experiments, the IT++ library[1] version 4.2 has been employed. IT++ is a C++ library mainly used in analysis and simulation of communications systems, developed and maintained by a community of industry and academic researchers. IT++ can also use and interoperate with other existing open-source libraries. We linked the code with ATLAS[2] (Automatically Tuned Linear Algebra Software), a software package for linear algebra offering an implementation that is at the same time efficient and portable. Routines in ATLAS are written to be competitive with machine-specific versions of the original BLAS (Basic Linear Algebra Subprograms) for a variety of architectures, and try to achieve good performance also on unknown machines.

As discussed in Chapter 4.3, the FastICA algorithm has two variants, corresponding to different approaches to orthonormalization, both of which are implemented in the IT++ library. The deflation approach estimates the components one after another while ensuring that orthogonality is maintained, whereas the symmetric approach computes the updates in parallel and applies symmetric orthogonalization to the de-mixing matrix at each iteration. The library also provides two different types of nonlinearity: the kurtosis (codenamed POW3) and the hyperbolic tangent (tanh). The choice of the nonlinearity significantly affects performance. In our tests, we used the symmetric approach and kurtosis nonlinearity.

The Waikato Environment for Knowledge Analysis (WEKA) [1] has been used to

[1] http://itpp.sourceforge.net/devel/index.html

[2] http://math-atlas.sourceforge.net/

build the tree-based binary classifier based on the J48 Java-based implementation of C.4.5. Since decision tree-based algorithms such as C4.5 can sometimes result in excessively large trees, overfitting the training data, J48 extensively uses tree pruning as a method to cope with the risk of overfitting, by relaxing the specificity of the decision tree in order to improve its performance on the testing set. In detail, J48 gradually generalizes a decision tree until it gains a balance of flexibility and accuracy by means of two pruning methods, respectively known as subtree replacement (reducing the number of tests along a path by replacing decision nodes with leafs) and subtree raising (moving nodes towards the root, by replacing other nodes along the way).

To achieve a more realistic evaluation and ensure maximum reliability of the whole training process, we post-processed the final results through 10-fold cross validation on all the training set constituents. To do this, training data is randomly partitioned into ten—mutually exclusive—blocks of approximately equal size, and the resulting classifier is tested on each one of the ten blocks, after having been trained on the remaining nine. The cross-validation estimation of accuracy results from the ratio between correct classifications, and the number of samples in the training data. In order to further validate our results, we compared the performance of J48 with two other well-known classification algorithms available in WEKA: BayesNet and OneR.

BayesNet is an implementation of Bayesian networks [2]. A Bayesian Network over a set of variables X is a directed acyclic graph (DAG) whose vertices correspond to the elements of X, together with a Conditional Probability Table (CPT), describing the probability of an element x given its parents in the DAG. BayesNet make explicit the idea that classification learning fundamentally amounts to estimating class probabilities, which are more expressive of plain predictions, for they also express the confidence in classification. More precisely, what is sought is an estimate of the conditional probability of the class given the values of the attributes. The classification task is to find an appropriate Bayesian Network starting from a dataset consisting of pairs (x, y) where y is the class variable, specifying the class of

x.

Different Bayesian networks can be constructed to represent the same probability distribution. Machine learning techniques for Bayesian networks induce a network from the correlations in the data. Learning is done in two stages: first, the network structure is learnt, exploring the space of possible networks; then, the probability tables are populated by computing the relative frequencies of the matching combinations of attribute values. Learning the structure corresponds to learning the edges, for nodes correspond to the attributes (including the class attribute). An approach for building a network structure is to maximize a quality measure of the structure, given the training data. Quality is gauged by the probability of the data, given the network. In practice, log-likelihood is preferred in place of probability, because the probabilities involved in the CPTs are usually very small. Maximization of the log-likelihood of the network, given the training data, can lead to overfitting if applied without correctives. In fact, there always is an incentive to add a new edge to the network. A common strategy to combat overfitting is to restrict to a predefined limit the maximum number of edges going out of each node, or the number of edges in the entire network. Alternatively, a penalty term can be added to limit the complexity of the network. For example, the AIC score, a measure for evaluating the quality of a network based of the Akaike Information Criterion (AIC), combines the negative log-likelihood with the number of free parameters:

$$A = -L + K \tag{6.1}$$

where A is the AIC score, L the log-likelihood, and K the number of parameters. Clearly, the lower is the AIC score, the better is the network. To find the number of parameters, it should be noted that, in a CPT, the last column is determined by the values in the other columns because probabilities must sum to 1. Therefore, the number of parameters is the number of independent entries in the CPTs. An important propriety is that metrics such as the AIC score are local, i.e., thay can be optimized separately for each node. The penalty term is evidently local, and the log-likelihood is a sum where terms can be rearranged and grouped so that each

term depends only on the entries in a single CPT. Thus, considering one node at a time, edges to other nodes can be added or removed, evaluating the result, provided that the network remains acyclic. This can be achieved, for example, by analyzing the nodes in a predefined order and only considering as potential parents the nodes which have already been visited. Because the results will depend on the initial order, other methods have been proposed, including simulated annealing, tabu search, and genetic algorithms.

OneR [3] is a simple but effective rule-based classifier, belonging to a family of classifiers, called *1R*, that classify samples on the basis of a single attribute. The OneR algorithm takes a training set as input and produces a set of rules—all based on a particular attribute—by ranking attributes in accordance to the error rate. Despite being simple, 1R classifiers perform well on a wide variery of data, and they remarkably cater for missing values in a straightforward way.

Before discussing the results, some considerations about systematic techniques to evaluate anomaly detection methods and compare different methods with one another, as well as about the datasets, are in order.

6.1 Classification Performance Metrics

To evaluate the classification power of anomaly detectors, metrics commonly used in machine learning are often used [4]. Measuring the performance of a classifier and, thus, being able to compare it with competing approaches is a fundamental, but by no means easy, task in machine learning. In particular, it is important to be able of assessing the cost of misclassification, taking the type of error into account.

The confusion matrix $\mathbf{C}$ is a contingency table detailing the frequencies of the events "one instance belonging to class i is classified as belonging to class j". Entry c_{ij} (see Fig. 6.1) is, thus, the count of items whose true class is i that were classified as belonging to class j. In the anomaly detection context, the classification is binary. Suppose that the two classes, normal and anomalous, are labeled, respectively with

		Classified as	
		Negative	Positive
Actual Class	Negative	*TN*	*FP*
	Positive	*FN*	*TP*

Figure 6.1: Confusion matrix for a binary classification task.

Negative (N), and Positive (P). The four classification outcomes depending on the true class are the classical ones:

- True Negatives (TN). Count of normal elements classified as normal.
- False Positives (FP). Count of normal elements classified as anomalous.
- False Negatives (FN). Count of anomalous elements classified as normal.
- True Positives (TP). Count of anomalous elements classified as anomalous.

Exploratory analysis of the confusion matrix can be used for performance evaluation of classifiers. Classic indicators are the *sensitivity* or True Positive Rate and the *specificity* or True Negative Rate:

$$TPR = \frac{TP}{TP + FN} \qquad TNR = \frac{TN}{TN + FP} \quad . \tag{6.2}$$

The sensitivity gives the proportion of actual positives that were correctly identified as positives (anomalies). Correspondingly, the specificity represents the fraction of actual negatives that were correctly identified as negatives. In information retrieval, the usual performance measures are *precision* and *recall*. The precision is the proportion of predicted positives that are actually positive, while the recall is the same as the sensitivity described above.

$$precision = \frac{TP}{TP + FP} \qquad recall = \frac{TP}{TP + FN} \quad . \tag{6.3}$$

An ideal indicator would summarize all the information in the confusion matrix into a single number, so that a simple comparison may be used for ranking classifiers.

Accuracy, the fraction of correctly classified instances, is a single-figure criterion in widespread use:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{6.4}$$

whereas its specular measure is the error rate:

$$error\_rate = 1 - accuracy \quad . \tag{6.5}$$

Although accuracy is an intuitive metric, it has, however, been criticized [5] as not taking the type of misclassification errors into proper account. The *F-measure* is the harmonic mean of precision and recall:

$$F\text{-}measure = \left(\frac{1}{precision} + \frac{1}{recall} \right)^{-1} . \tag{6.6}$$

Receiver-Operating-Characteristic (ROC) analysis, and in particular the Area Under the ROC Curve (AUC) [6], is a method for evaluating data mining methods that derives from the trade-off, usually met in signal detection, between hit rate and false alarm rate on noisy channels. Assuming a threshold-based classifier, the ROC curve is a plot of the sensitivity versus the specificity as the threshold varies. Optimal classification would correspond to the upper left-hand corner. Alternatively, the ROC curve can be drawn by considering the actual positive instances sequentially and marking on the x (resp., y) axis the points corresponding to the total count of false positives (resp., true positives) found so far, divided by the total number of actual positives. A sample ROC curve obtained with the latter method is shown in Fig. (6.2). To avoid the dependence on the order of test data, commonly used solutions entail the ranking of instances in some way. Ranking can be generated on the basis of the confidence of classification, where available, or by means of the results of a number of cross-validations. Again in the quest for a single quantity that could summarize the information in the ROC curve, the AUC has been introduce as a measure of quality, on the observation that better models tend to correspond to larger areas.

The Matthews Correlation Coefficient (MCC) [7] measures the statistical correlation between actual and predicted values. Put differently, it is a measure of the

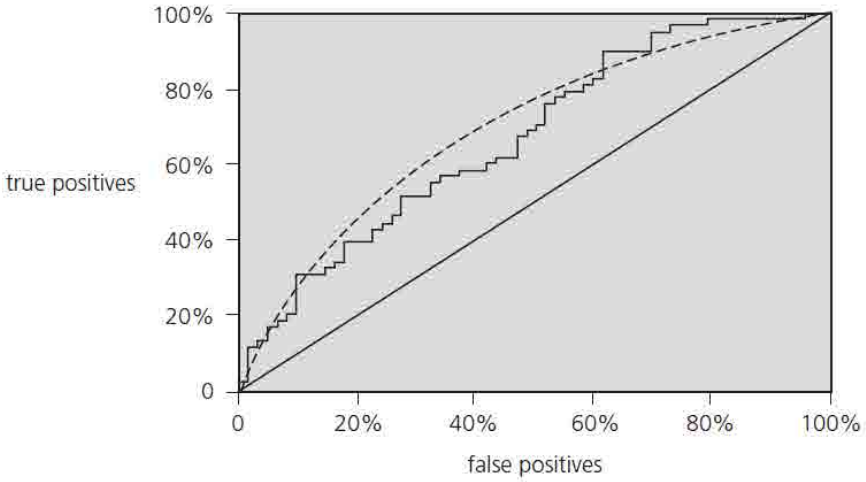

Figure 6.2: A sample ROC curve [5].

linear association between truth and prediction, calculated as the Pearson product-moment correlation coefficient:

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{6.7}$$

The MCC lies in the interval $[-1, 1]$, with a value of 0 indicating no correlation, a value of -1 indicating completely wrong classification (which, for binary classification, only needs reversal) and a value of 1 indicating perfect classification. MCC has been recommended [8] as an indicator exhibiting a good trade-off between consistency and discriminatory ability. However, when the number of positives (i.e., TP + FP) is low, the MCC will be very high in proportion.

Cohen's Kappa indicator statistic measures the agreement between predictions and actual labels, correcting for the agreements that might have occurred by chance. The Kappa indicator κ is computed as

$$\kappa = \frac{P_a - P_r}{1 - P_r} \tag{6.8}$$

where P_a is the probability of correct classification (estimated by the accuracy) and P_r is the probability for a random classificer to produce an accurate classification,

estimated by

$$P_r = \frac{(TP+FP)(TP+FN)+(TN+FP)(TN+FN)}{(TP+TN+FP+FN)^2}\,. \tag{6.9}$$

In other terms, the indicator expresses the proportion of classification success with respect to the performance of a perfect classifier.

Other indicators were proposed which have their theoretical ground in Information Theory [9]. The Confusion Entropy (CEN) [10].

$$\begin{aligned} CEN &= \frac{(FN+FP)\log_2\left((TP+TN+FP+FN)^2-(TP-TN)^2\right)}{2(TP+TN+FP+FN)} \\ &\quad -\frac{FN\log_2 FN + FP\log_2 FP}{TP+TN+FP+FN} \end{aligned} \tag{6.10}$$

In [11], the similarity between the MCC and the CEN is studied, showing that the two measures are strongly correlated. Gu *et al.* also proposed a metric for the evaluation of intrusion detection systems: the Intrusion Detection Capability C_{ID} is the mutual information between the IDS input and output, normalized with respect to the entropy of input. The C_{ID} is

$$C_{ID} = \frac{I(X;Y)}{H(X)} \tag{6.11}$$

where X and Y are two random variables associated with the input and output respectively, $I(X;Y)$ is their mutual information and $H(X)$ the entropy of X. Recall that, in information theory, entropy is a measure of uncertainty associated with a random variable. If X is a discrete random variable and $p(x)$ its distribution, its entropy $H(X)$ is defined as the expectation of the information

$$H(X) = E\left\{\log\frac{1}{p(x)}\right\} = -\sum_x p(x)\log p(x)\,. \tag{6.12}$$

Different units of entropy are obtained, depending on the basis of the logarithm. If $p(x,y)$ is the joint distribution of X and Y and $p(y|x)$ the conditional distribution of Y given X, the conditional entropy $H(Y|X)$ is defined as

$$H(Y|X) = -\sum_x\sum_y p(x,y)\log p(y|x) \tag{6.13}$$

with the interpretation of $H(Y|X)$ being the degree of uncertainty remaining about Y once X is determined. The mutual information $I(Y;X) = I(Y;X)$ is a symmetric quantity that can be interpreted as giving the reduction of uncertainty in one of the random variables once the other is known. It can be calculated as

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)\,. \tag{6.14}$$

6.2 Testing on Real Traffic Data

Standardized datasets for testing would be of great value for achieving reliable tests of a newly developed anomaly detection system, and for comparing its performance to other algorithms. Nevertheless, publicly available datasets are rare. Network administrators are reluctant to provide network packet captures to the research community. Network traces contain indeed sensitive information, some of which can be extracted even after that attempts at masking it are brought into action: extensive research has been dedicated to anonymization and de-anonymization of packet traces. For example, Pang *et al.* [12] discuss the conflicting goals of trace anonymizers, underlining that these systems should confuse data to the greatest possible extent but without destroying important information that may be relevant to researchers. On the other hand, despite being relatively simple to produce, synthetic traces might not adequately reflect all the complexity of real-world traffic.

The DARPA/Lincoln Labs and the KDD99 datasets, though old and deprecated, continue to be widely used in assessing the performance of anomaly detection algorithms. The classical DARPA/Lincoln Labs dataset contains labeled network traces for several days from a simulated network [13][14]. The DARPA dataset was in fact criticized [15] as being synthetic, thus not reflecting real-world traffic characteristic, for both usage patterns and actual attacks. Ideally, a dataset should show statistically verifiable similarity with typical traffic captured in the network within regular operation. In the same vein, attacks can be expected to span all vulnerable hosts,

and not be limited to a subset of them. The KDD cup dataset [16] was derived from DARPA traces, extracting the essence of them into a set of features. It could not possibly, therefore, avoid the shortcomings of the DARPA dataset. In addition, it raised criticism as per the rates of some events [17], which were frowned on as being too uniform. The KDD dataset was extensively analyzed in [18], finding that a large number of redundant records are contained within it. This causes a bias in the learning algorithms towards the more frequent attacks. The NSL-KDD is an improvement over the KDD dataset. The NSL-KDD includes a selection of records from the complete KDD dataset and solves the issues raised in [18]. Redundant records have been removed from the NSL-KDD training dataset, and the number of records for each group is more balanced than in the original KDD dataset. However, NSL-KDD still suffers from the shortcomings detailed in [15]. Another publicly available data set is the LNBL dataset, that contains enterprise traffic anonymized with the tcpmkpriv tool [12]. Traces only include packet headers, without payload. In addition, attack traffic filtered at the border is not available, as traffic was captured for two internal subnets.

An important issue that should be considered is the extent to which test datasets retain their significance over time. In fact, the usage of networks is constantly evolving, often in radical ways. Consequently, new applications emerge and usage patterns change, sometimes dramatically. As a result, there is no guarantee that a baseline which used to be valid years before will still reflect the characteristic of current normal traffic. Even though a complete dataset was available at a certain moment, its validity as a test tool would fade over time.

6.3 Analysis of the Results

The training and testing traces have been captured by using 5 different sensors, located on independent workstations operating within the same LAN segment, with a sampling rate of 1 second, properly chosen to cover some typical cases such as the

noticeable differences in network usage between morning and evening hours.

Table 6.1: General workload dimensions of the traces.

Trace	Training Set	Testing Set
Duration	24 hours	24 hours
Anomalous Samples	26400	3900
Packets	$5.82 \cdot 10^8$	$6.67 \cdot 10^8$
Bytes	$2.27 \cdot 10^{11}$	$2.18 \cdot 10^{11}$

The traces contain several anomalous events simulated through distributed SYN floods occurring at various times (see Tables 6.2 and 6.3). A SYN flood attack is a very well known attack that consists in forcing the creation of many "half-open" TCP connections. By "half-open" it is meant that the attacker sends a TCP packet with the SYN flag set, to which the server responds with a packet having the SYN and ACK flags set. The server will now be waiting for the final ACK packet to complete the connection setup, but the attacker would never send that packet. A great number of resources will be spent on the server side to keep track of these unfinished connection setups. Address spoofing allows the attacker to send multiple requests without disclosing its real address, for the attacker doesn't care to actually establish a connection and can safely ignore the server reply. Several variations exist that are based on this basic setting. Such variations involve other aspects of the TCP protocol, with the use of other flags, or taking advantage of other machines as reflectors, by spoofing the address of the victim in requests sent to the reflector, thus forcing it to send reply packets to the victim. In the distributed version of this attack, many machines join their attacking effort onto a single target.

The number and duration of the anomalies in the training set is considerably higher than in the testing set, to provide the classifier with a sufficiently rich set of anomalies.

The above three types of attacks have been chosen both for duration and specific characteristics (i.e., their explicit "noisy" behavior) to be a sufficiently consistent and

Table 6.2: Simulated attacks in the training traces.

Training Set			
Start Time	**Duration**	**Attack**	**Rate**
00:00	190min	SYN flood	500/s
06:00	15min	SYN flood	500/s
09:00	20min	SYN flood	500/s
13:00	195min	SYN flood	250/s
19:00	20min	SYN flood	250/s

representative sample for a volume-based analysis. Most of the anomalous traffic patterns that can be currently observed on the Internet (inbound DDoS attacks, bandwidth floods, single and multiple scans) can be associated to these attack types.

After processing the raw traffic data with the CoralReef suite, the FastICA algorithm has been run on the resulting data in order to separate the independent components and determine the unmixing matrix. For example, the computed unmixing matrix for the testing data (limited, for the sake of presentation clarity, to the packet counts) is reported in Fig. 6.3.

$$W = \begin{pmatrix} -0.001629 & 0.000026 & 0.000004 & -0.000006 & 0.000019 \\ 0.000431 & -0.000532 & -0.001766 & 0.000023 & -0.000374 \\ 0.000748 & 0.002665 & -0.000490 & -0.000563 & 0.001420 \\ 0.000116 & -0.000058 & 0.000284 & -0.000094 & -0.000364 \\ 0.000087 & -0.000339 & 0.000259 & 0.000096 & 0.000579 \end{pmatrix}$$

Figure 6.3: Unmixing matrix for the testing data (packet counts only).

Some of the coefficients are relatively close to zero, indicating an almost complete independence of the corresponding signal/component pair. Other elements show

Table 6.3: Simulated attacks in the testing traces.

Testing Set			
Start Time	**Duration**	**Attack**	**Rate**
02:00	5min	SYN flood	500/s
05:00	5min	SYN flood	250/s
08:00	10min	SYN flood	500/s
14:00	15min	SYN flood	500/s
17:00	5min	SYN flood	250/s
20:00	10min	SYN flood	250/s
23:00	15min	SYN flood	500/s

a non-negligible dependence. As stated before, neither the sign nor the absolute values are significant, on account of the inherent indeterminacy of the separation into independent components.

6.3.1 ICA at a single collection point

The first analysis is meant to answer the question of whether and to what extent the counts of packets and bytes taken at a single location in the network are related. For readability of the graphs that will be shown next, only two features have been selected. In particular, the choice privileged packet and byte counts because these features are most prominent and convey much information. The scatter plots in Fig. 6.4, obtained by analyzing the packet and byte counts at a single capture point, confirms the intuition that these counts are strongly interdependent. Plots on symmetrical positions in the figure are the same plot with axes swapped, while the straight lines in the diagrams on the secondary diagonal are scatter plots of each signal against itself. The diagram in the upper right corner of Fig. 6.4, in fact, diverges significantly from a straight line, showing three clusters of points. The

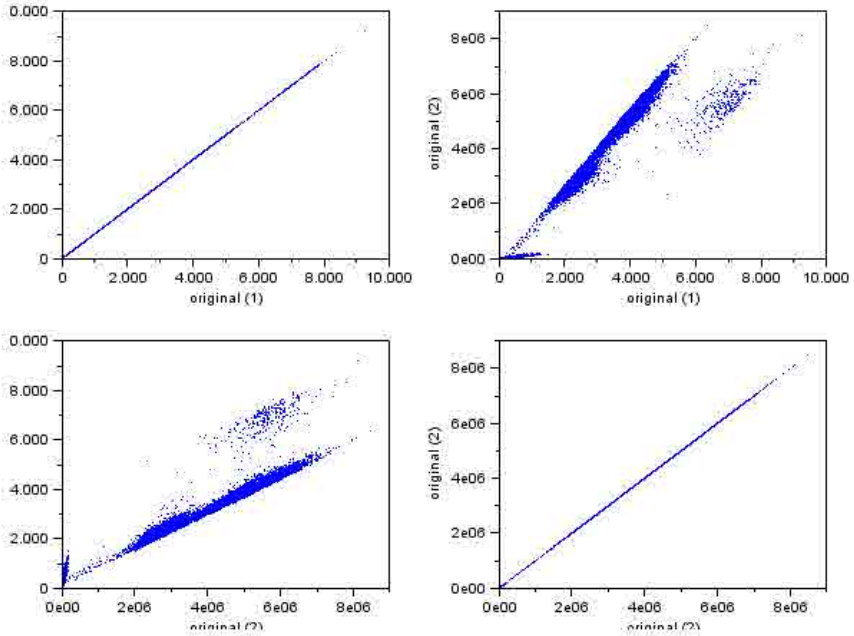

Figure 6.4: Scatter plot of packet and byte counts.

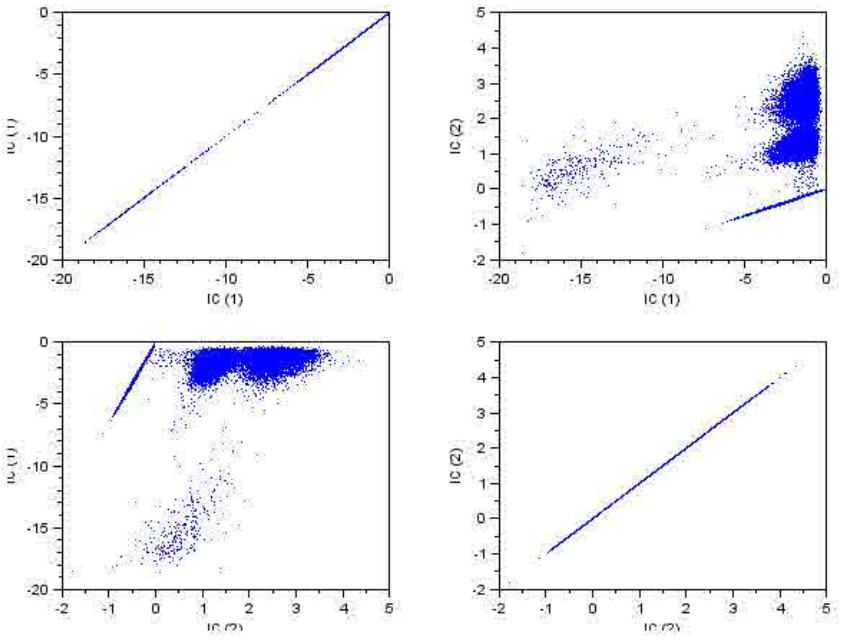

Figure 6.5: Scatter plot of the independent components (IC).

cluster near the origin is related to small packets, while the other two describe the majority of packets and a small group of relatively large packets.

After performing the separation into independent components, it is natural to examine the mutual relationship between the obtained components. Analogously to Fig. 6.4, Fig. 6.5 contains the scatter plots of the independent components, one versus the other. As it can be seen in Fig. 6.5, graph at the top right, there still remains some regularity (the straight line), but the majority of points are spread across a wider area than in Fig. 6.4. In addition, there are at least four observable clusters instead of three, suggesting a more detailed structure in the independent components and in the relationship between them. Looking in detail at the rela-

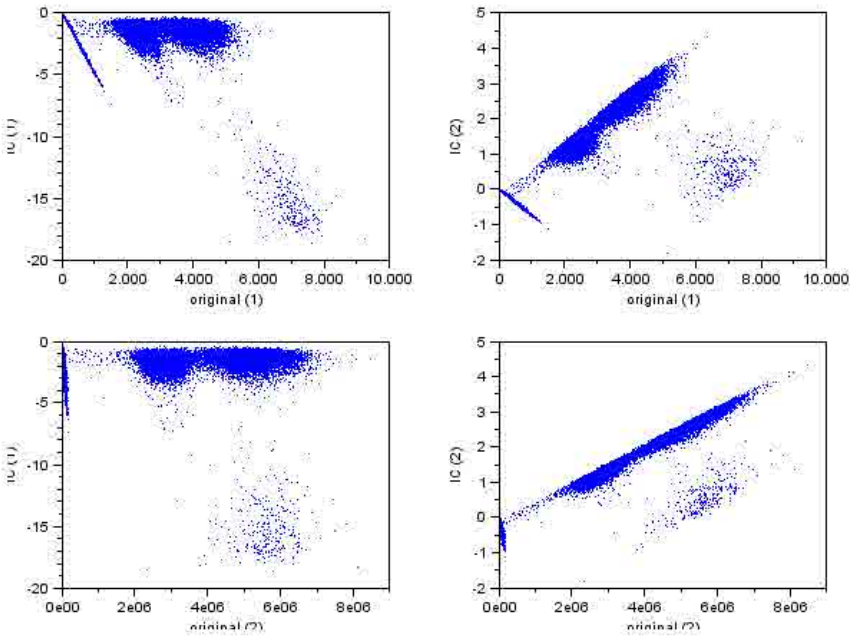

Figure 6.6: Scatter plot of packet and byte counts vs. the ICs.

tionship between the original signals (the packet and byte counts, labeled as 1 and 2) and the independent components, shown in the scatter plots in Fig. 6.6, it is evident that the second component is strongly coupled with the byte count (Fig. 6.6, bottom right) and with the packet count (Fig. 6.6, top right). At the same time, the first component has a more complex linkage with both of the original signals

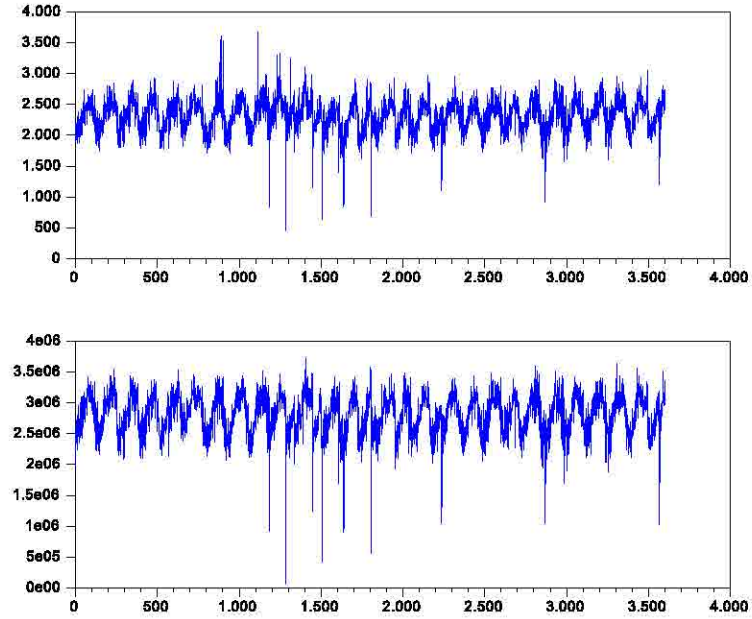

Figure 6.7: Time series chart of packets and bytes, single collection point.

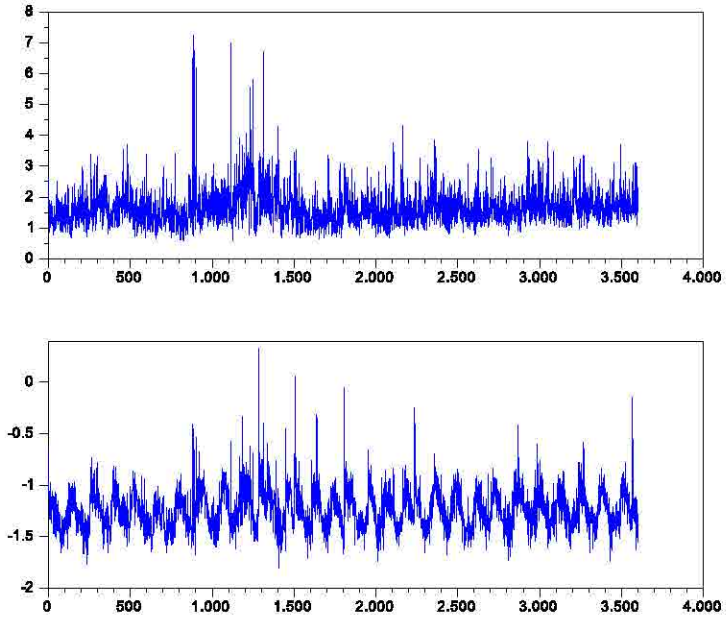

Figure 6.8: Time series chart of the ICs, single collection point.

(Fig. 6.6).

The time series chart of Fig. 6.7 is relative to the packet and byte counts measured at a single host for the time interval from 01:00 pm to 02:00 pm, while Fig. 6.8 contains the time series charts of the corresponding independent components for the same interval. Inspecting closely the time series charts in Fig. 6.7 and confronting them with the plots of the independent components in Fig. 6.8, we can see that the noticeable similarity between the original features is lost when the independent components are examined. While one of the independent components closely resembles the shape of the original signal, the other independent component has a different behavior, with its own spikes, making evident information that was less readily accessible in Fig. 6.7.

The previous plots confirm the intuition that observed data have much more structure in them than it is immediately apparent, and that ICA can be used to expose some of that structure, even when a single collection point is considered.

6.3.2 ICA at five collection points

The second set of experiments involved data captured at all five sensors. Again, simplified versions of the charts are reported to enhance readability.

Analogously to the previous figures showing the time series chart of the measurements and of the independent components, Figs. 6.9–6.12 report the time series charts for the packets and bytes as measured at the five collection points, as well as for the corresponding independent components, for the time interval from 07:00 am to 09:00 am. Figures 6.9 and 6.10 depict the packet counts and the independent components obtained from these packet counts in isolation, i.e., without considering other features when applying ICA. The same holds for Figs. 6.11 and 6.12, which refer to the byte counts. The charts are, thus, only indicative. Nevertheless, the comparison between Figs. 6.9 and 6.10 shows that a pattern of stepwise increasing activity, observable in at least two of the subplots in Fig. 6.9, is more compactly described by a more rise in a single component, namely the third one of Fig. 6.10,

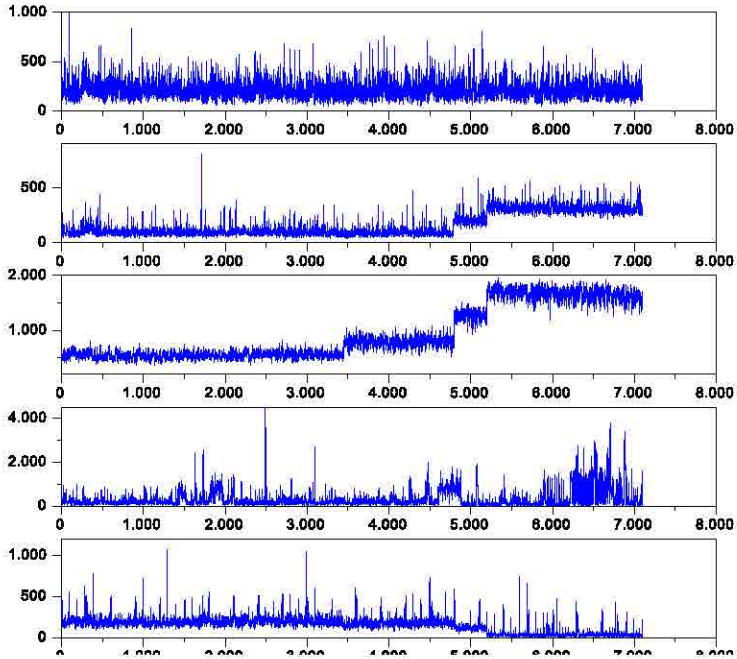

Figure 6.9: Time series chart of packets, five collection points.

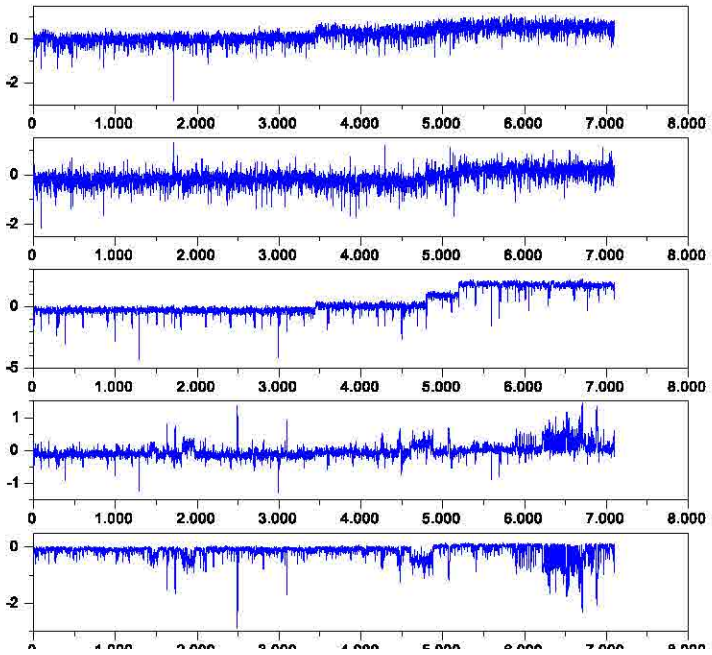

Figure 6.10: Time series chart of ICs from packets, five collection points.

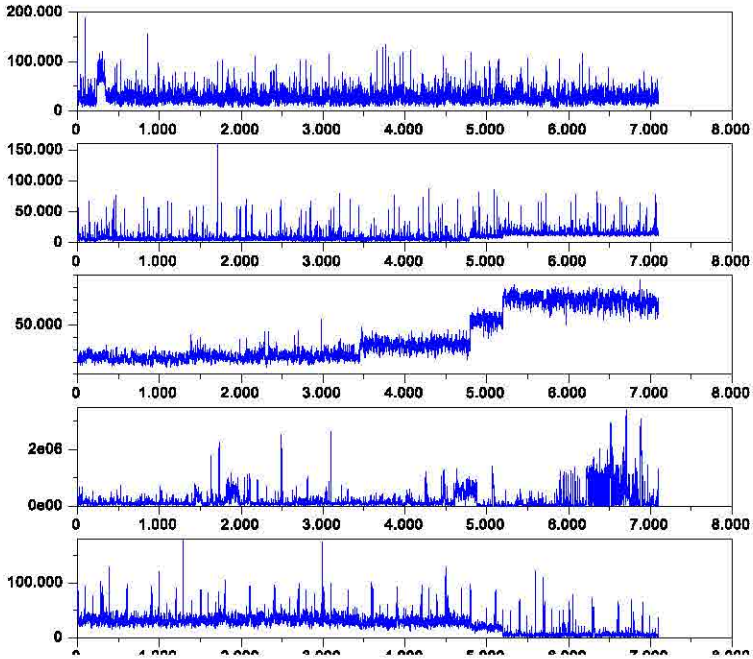

Figure 6.11: Time series chart of bytes, five collection points.

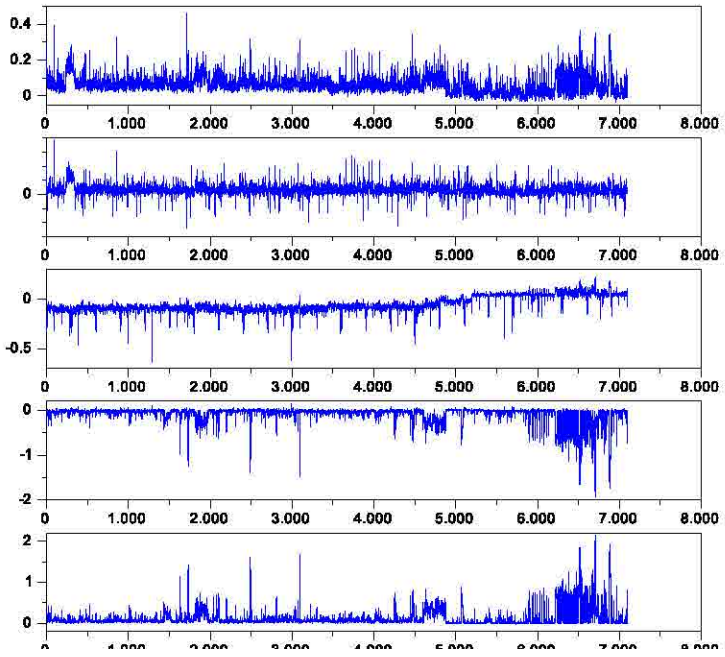

Figure 6.12: Time series chart of the ICs from bytes, five collection points.

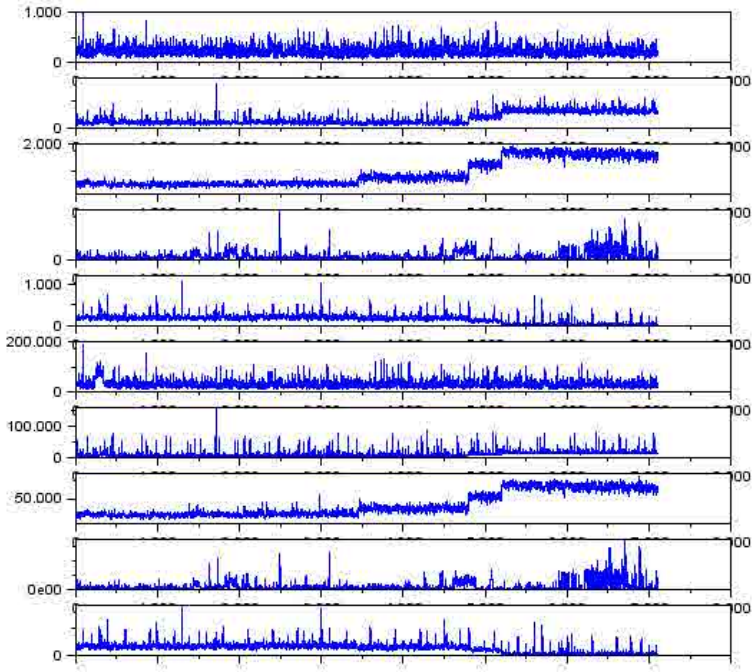

Figure 6.13: Time series chart of packets and bytes, five collection points.

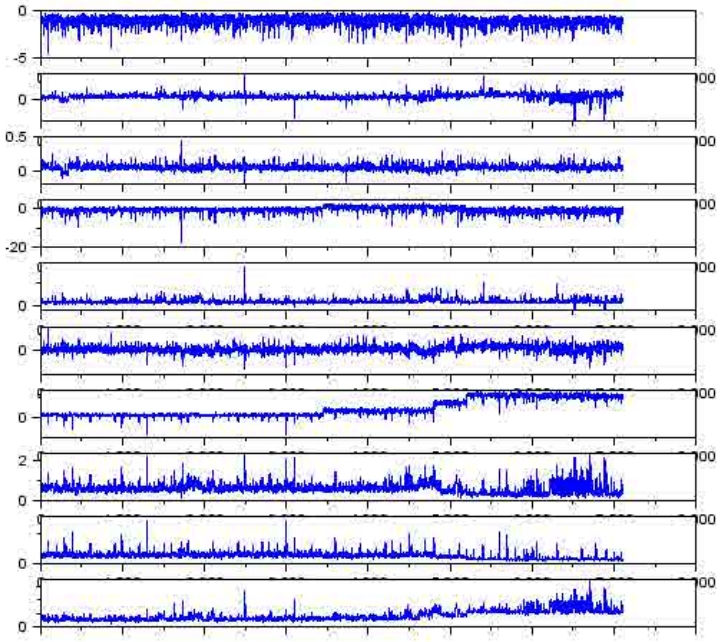

Figure 6.14: Time series chart of ICs from packets and bytes, five collection points.

allowing other components to describe other aspects.

Figures 6.13 and 6.14 show the results of the application of ICA to the combined set of packet and byte counts from all the five collection points. Again, the stepwise increase is evident in a single independent component, but there apparently is no other significant improvement in comparison with Figs. 6.9–6.12. This suggests that, since the features considered are strongly correlated, the application of ICA to a restricted set of them is sufficient to expose sone hidden structure. Such finding would be beneficial in a practical implementation, where limiting the transformation to a reduced set of features could be an important factor in improving the execution time of the anomaly detection system. Note that, in passing, this would also allow the analysis of longer time windows.

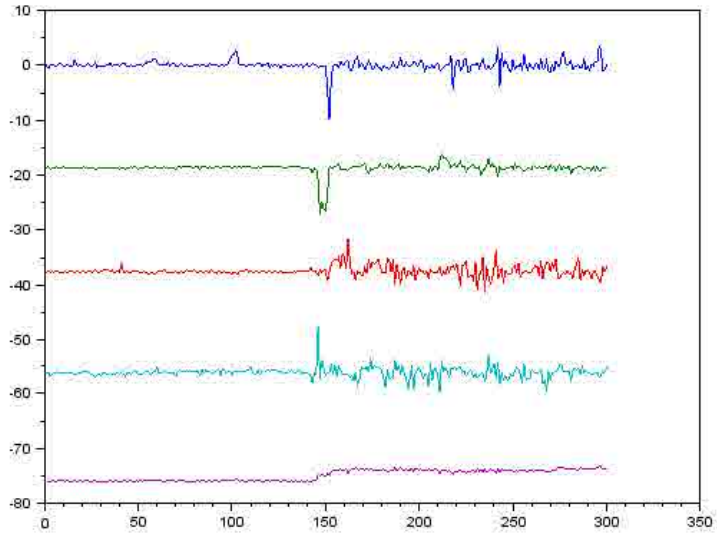

Figure 6.15: Time series chart of the independent components (IC) relative to pkts.

A closeup chart is shown in Fig. 6.15, which depicts the independent components (obtained from the packet counts only, at all locations) in a short time window that has been chosen so that it includes the beginning of an attack. It can be seen

that three components exhibit a clear spike in correspondence to the beginning of the attack. Interestingly, the timing is slightly different: not all components react simultaneously.

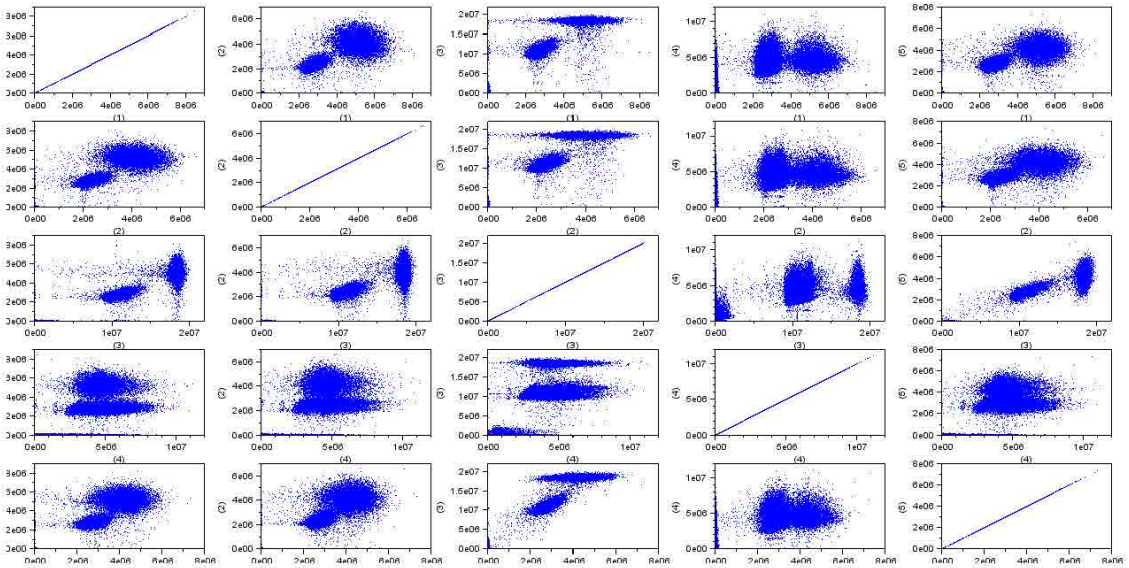

Figure 6.16: Scatter plot of the byte counts at the five collection points.

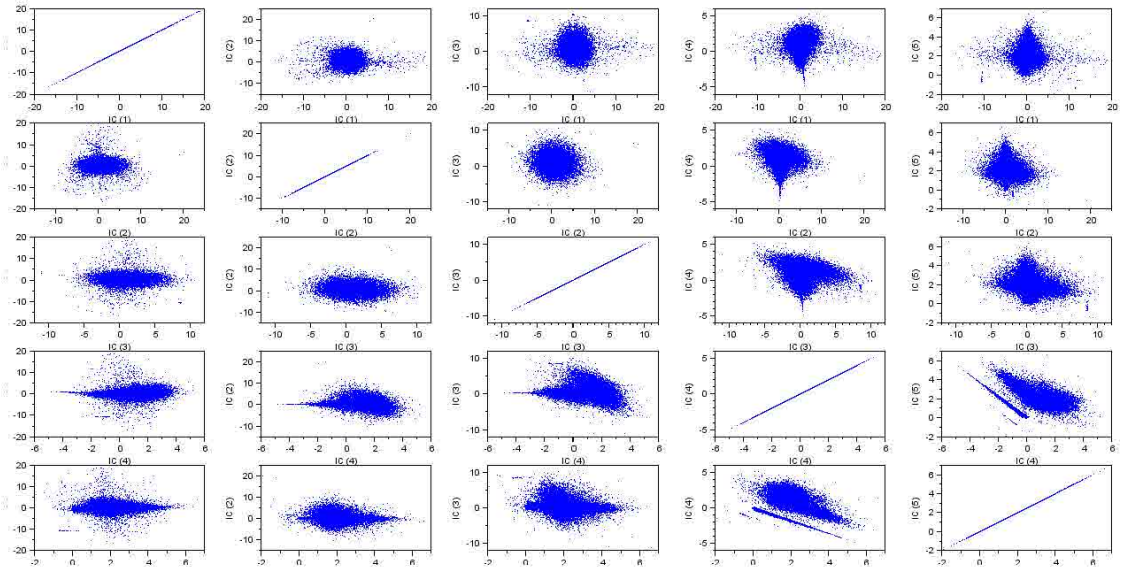

Figure 6.17: Scatter plot of ICs from bytes at the five collection points.

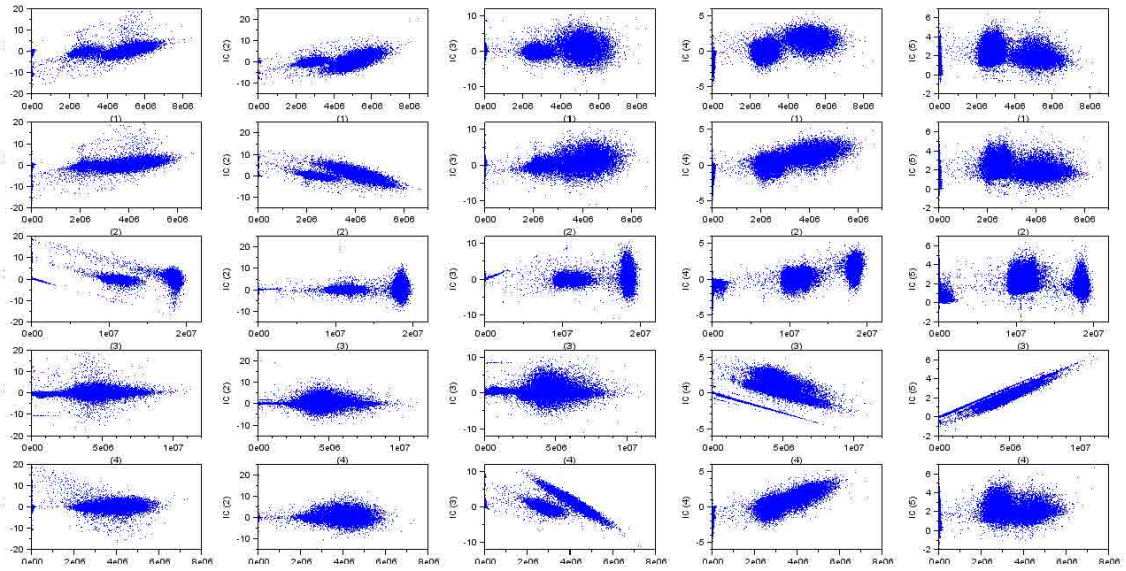

Figure 6.18: Scatter plot of bytes vs. ICs from bytes at the five collection points.

The scatter plots in Figs. 6.16, 6.17, and 6.18 depict, respectively, the relationship between the byte counts at the five locations, the independent components obtained by separating those byte counts, and the mutual relationship between the byte counts and the independent components. As stated earlier, the fact that the ICA procedure and the plots have been purposely restricted to a single feature, namely the byte count, collected at multiple points, is not restrictive of the meaningfulness of the plots, because the effects of separation are already evident in the simple plots and the complete set pf plots would have been unreadable. By comparing Fig. 6.16 with Fig. 6.17, the effectiveness of the separation procedure is immediately evident. Almost all the byte counts show a complex pattern of relationship with the byte count collected at a different location, resulting in two clusters of points noticeable on the scatter plots in Fig. 6.16. An exception is the (unordered) pair (3,4), where the relationship is even more complex, for there are three clusters. In contrast, as can be expected, the scatter plots in Fig. 6.17 have a much more concentrated, simpler structure. By analysis of the scatter plots in Fig. 6.18—which, differently from the ones in Figs. 6.16 and 6.17, are not symmetric—we can see roughly linear shapes in some diagrams, pointing to the similarity between the involved signal and

component. The best accordance is shown by that the fifth independent component and the fourth signal, which are strongly related, as can be also seen from the corresponding time series subplots in Figs. 6.11 and 6.12. In addition, the three clusters which were present in the scatter plots at subgraph (3,4) and its symmetrical in Fig. 6.16 are noticeable across all the scatter plots relative to the third signal, but not the fourth.

6.3.3 Performance of clasifiers

Table 6.4: Classifier Performance Comparison

Parameter	OneR	J48	BayesNet
Percent correct	93.72	99.38	89.28
Kappa statistic	0.534	0.926	0.393
TP rate	0.895	0.900	0.912
FP rate	0.061	0.002	0.108
TN rate	0.939	0.998	0.892
FN rate	0.105	0.100	0.088
TP	3491	3511	3556
FP	5018	143	8919
TN	77482	82357	73581
FN	409	389	344
Precision	0.410	0.961	0.285
Recall	0.895	0.900	0.912
F measure	0.563	0.930	0.434
Matthews correl.	0.927	0.94	0.475
Area under ROC	0.917	0.949	0.960

As to the performance of classifiers, the results summarized in Table 6.4 show that, among the classifiers tested, only the J48 classifier achieved a satisfactory performance. This is largely due to the percentage of false positives, which is signif-

Table 6.5: Classifier Performance Comparison with and without ICA

Measure	w/o ICA	with ICA
Percent correct	99.34	99.38
Kappa statistic	0.923	0.926
TP rate	0.894	0.900
FP rate	0.002	0.002
TN rate	0.998	0.998
FN rate	0.106	0.100
TP	3488	3511
FP	141	143
TN	82359	82357
FN	412	389
Precision	0.961	0.961
Recall	0.894	0.900
F measure	0.927	0.930
Matthews correl.	0.924	0.940
Area under ROC	0.937	0.949

icantly lower than the one obtained by the other classifiers. BayesNet has, by a slight slack, the highest number of true positives, but that is counterbalanced by the false positive rate, which is by far the highest among the three algorithms. In contrast to the wide variation in the tallies of false positives, it is interesting to note that all algorithms find a similar number of false negatives. Again, the lowest percentage is shown by BayesNet, with OneR consistently being the worst performer with a

false negative rate of 0.105. The consensus of the three different algorithms on small false negative rates is a confirmation of the effectiveness of ICA to isolate anomalous events from the traffic, so that the resulting components show a distinctive behavior that can be detected even by classifiers that do not perform particularly well under other respects.

Finally, Table 6.4 reports the comparison of the results for the best-performing classifiers (i.e., J48) when run on the observed data and on the transformed data by means of ICA. It can be seen that the performance enhancement is small but consistent across the metrics, with the most notable improvements an increased number of true positives and a decreased number of false negatives.

In conclusion, the preliminary results are interesting and indicate that the proposed method is a promising avenue for further investigation.

References

[1] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten, "The WEKA data mining software: an update," *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[2] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Machine learning*, vol. 29, no. 2, pp. 131–163, 1997.

[3] R. Holte, "Very simple classification rules perform well on most commonly used datasets," *Machine learning*, vol. 11, no. 1, pp. 63–90, 1993.

[4] T. Fawcett, "An introduction to ROC analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[5] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[6] J. A. Hanley and B. J. McNeil, "The meaning and use of the area under a receiver operating characteristic (ROC) curve.," *Radiology*, vol. 143, no. 1, pp. 29–36, 1982.

[7] B. W. Matthews, "Comparison of the predicted and observed secondary structure of t4 phage lysozyme," *Biochimica et Biophysica Acta (BBA)-Protein Structure*, vol. 405, no. 2, pp. 442–451, 1975.

[8] P. Baldi, S. Brunak, Y. Chauvin, C. A. Andersen, and H. Nielsen, "Assessing the accuracy of prediction algorithms for classification: an overview," *Bioinformatics*, vol. 16, no. 5, pp. 412–424, 2000.

[9] G. Gu, P. Fogla, D. Dagon, W. Lee, and B. Skorić, "Measuring intrusion detection capability: an information-theoretic approach," in *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, ACM, 2006, pp. 90–101.

[10] J.-M. Wei, X.-J. Yuan, Q.-H. Hu, and S.-Q. Wang, "A novel measure for evaluating classifiers," *Expert Systems with Applications*, vol. 37, no. 5, pp. 3799–3809, 2010.

[11] G. Jurman, S. Riccadonna, and C. Furlanello, "A comparison of MCC and CEN error measures in multi-class prediction," *PloS one*, vol. 7, no. 8, e41882, 2012.

[12] R. Pang, M. Allman, V. Paxson, and J. Lee, "The devil and packet trace anonymization," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 29–38, 2006.

[13] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, *et al.*, "Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation," in *Proceedings, DARPA Information Survivability Conference and Exposition. DISCEX'00*, IEEE, vol. 2, 2000, pp. 12–26.

[14] R. Lippmann, J. Haines, D. Fried, J. Korba, and K. Das, "The 1999 DARPA off-line intrusion detection evaluation," *Computer Networks*, vol. 34, no. 4, pp. 579–595, 2000.

[15] J. McHugh, "Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory," *ACM transactions on Information and system Security*, vol. 3, no. 4, pp. 262–294, 2000.

[16] *KDD99*, http://kdd.ccs.uci.edu/databases/kddcup99/task.html.

[17] M. V. Mahoney and P. K. Chan, "An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection," in *Recent Advances in Intrusion Detection*, Springer, 2003, pp. 220–237.

[18] M. Tavallaee, E. Bagheri, W. Lu, and A.-A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications*, 2009.

Chapter 7

Conclusions

Accurately detecting anomalies in network traffic is effective to preserve good working conditions for the network, as well as increase availability. Timeliness is also important, as the time window for some attacks can be short. A good representation in term of features is a key factor to achieve a relatively compact model of normal traffic behavior, so that the individuation of abnormal conditions can be simplified. Flexibility and adaptivity in achieving such compact representations is needed, because traffic keeps changing, and a representation which is good at present might no longer yield good results years from now. In this context, we have analyzed the effects of a change of representation based on blind source separation techniques. This change of coordinates, coupled with off-the-shelf classifiers in a two-stage system, has been shown to improve the accuracy of classification on real traffic data. In particular, the proposed strategy has been designed with the aim of being able to "notice" the occurrence of suspicious activity when observing from multiple sources. The notion of statistical independence and the use of higher-order statistics is also promising in the quest of finding a more "natural" representation, closer to the unknown explanatory causes that determine the dynamics of traffic data-generating process.

It should be noted that the proposed scheme assumes that the target phenomena involve measurable variations of the statistical profile of the traffic time series. Thus, it cannot detect anomalous behaviors affecting only the payload of packet/transactions (e.g., buffer overflow or other vulnerability exploits). Analo-

gously, it can be less effective with attacks designed to be indistinguishable, from the point of view of the statistical properties, from regular activity.

The avenues for further research are multifarious. A main objective for future study is the pursuit of obtaining a good representation in a mostly unsupervised way, by means of deep learning mechanisms. In particular, the generative abilities of deep graphical model, coupled with the composition of denoising autoencoders, is a good candidate for anomaly detection. Preliminary steps in this direction have been taken with the analysis of the potential of RBM in this endeavor [1].

An interesting application of the main idea proposed here involves the detection of the specific traffic related to the command and control (C&C) channel of botnets. The intuition is that, since C&C traffic is strongly correlated temporally [2], ICA can isolate it in a single independent component.

Besides recognizing the effect of a phenomenon measured over a set of different sensing points, separation into independent components can be used also in another context. This application relates to the change of representation induced by ICA for a set of measurements taken at a single sensing point. An ongoing scientific investigation uses the separation of independent components within the flows sampled at a single collection point for the purpose of traffic characterization and application recognition, a problem that has received much attention in the literature [3]. The different independent sources can be estimated and their statistical characteristics can be associated, by means of a variety of methods, with distinct traffic flows connected with specific applications.

References

[1] U. Fiore, F. Palmieri, A. Castiglione, and A. De Santis, "Network anomaly detection with the Restricted Boltzmann Machine," *Neurocomputing*, vol. 122, pp. 13–23, 2013.

[2] G. Gu, R. Perdisci, J. Zhang, W. Lee, *et al.*, "Botminer: clustering analysis of network traffic for protocol-and structure-independent botnet detection.," in *USENIX Security Symposium*, 2008, pp. 139–154.

[3] F. Palmieri and U. Fiore, "A nonlinear, recurrence-based approach to traffic classification," *Computer Networks*, vol. 53, no. 6, pp. 761–773, 2009.

Appendix A

Log likelihood in RBMs

The derivative of the free energy (please refer to Chapter 3 for the notation) with respect to the parameter can be written as

$$\begin{aligned}
\frac{\partial \mathcal{F}(\mathbf{v})}{\partial \theta} &= -\frac{\partial}{\partial \theta} \log \sum_{\mathbf{h}} \exp(-\mathcal{E}(\mathbf{v}, \mathbf{h})) \\
&= -\frac{1}{\sum_{\mathbf{h}} \exp(-\mathcal{E}(\mathbf{v}, \mathbf{h}))} \frac{\partial}{\partial \theta} \sum_{\mathbf{h}} \exp(-\mathcal{E}(\mathbf{v}, \mathbf{h})) \\
&= -\frac{1}{\sum_{\mathbf{h}} \exp(-\mathcal{E}(\mathbf{v}, \mathbf{h}))} \sum_{\mathbf{h}} \frac{\partial}{\partial \theta} \exp(-\mathcal{E}(\mathbf{v}, \mathbf{h})) \\
&= \frac{1}{Zp(\mathbf{v})} \sum_{\mathbf{h}} \exp(-\mathcal{E}(\mathbf{v}, \mathbf{h})) \frac{\partial \mathcal{E}(\mathbf{v}, \mathbf{h})}{\partial \theta} \\
&= \sum_{\mathbf{h}} \frac{\exp(-\mathcal{E}(\mathbf{v}, \mathbf{h}))}{Z} \frac{1}{p(\mathbf{v})} \frac{\partial \mathcal{E}(\mathbf{v}, \mathbf{h})}{\partial \theta} \\
&= \sum_{\mathbf{h}} \frac{p(\mathbf{v}, \mathbf{h})}{p(\mathbf{v})} \frac{\partial \mathcal{E}(\mathbf{v}, \mathbf{h})}{\partial \theta} \\
&= \sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}) \frac{\partial \mathcal{E}(\mathbf{v}, \mathbf{h})}{\partial \theta} \\
&= E_{\mathcal{D}} \left\{ \frac{\partial \mathcal{E}(\mathbf{v}, \mathbf{h})}{\partial \theta} \right\} . \qquad \text{(A.1)}
\end{aligned}$$

Theferore, the derivative of the log likelihood is

$$\begin{aligned}
\frac{\partial \log p(\mathbf{v})}{\partial \theta} &= -\frac{\partial \mathcal{F}(\mathbf{v})}{\partial \theta} - \frac{\partial \log Z}{\partial \theta} \\
&= -\frac{\partial \mathcal{F}(\mathbf{v})}{\partial \theta} - \frac{1}{Z} \frac{\partial Z}{\partial \theta} \\
&= -\frac{\partial \mathcal{F}(\mathbf{v})}{\partial \theta} - \frac{1}{Z} \frac{\partial}{\partial \theta} \sum_{\mathbf{v}} \exp(-\mathcal{F}(\mathbf{v}))
\end{aligned}$$

$$
\begin{aligned}
&= -\frac{\partial \mathcal{F}(\mathbf{v})}{\partial \theta} - \frac{1}{Z}\sum_{\mathbf{v}} \frac{\partial \exp(-\mathcal{F}(\mathbf{v}))}{\partial \theta} \\
&= -\frac{\partial \mathcal{F}(\mathbf{v})}{\partial \theta} + \frac{1}{Z}\sum_{\mathbf{v}} \exp(-\mathcal{F}(\mathbf{v}))\frac{\partial \mathcal{F}(\mathbf{v}))}{\partial \theta} \\
&= -\frac{\partial \mathcal{F}(\mathbf{v})}{\partial \theta} + \sum_{\mathbf{v}} \frac{\exp(-\mathcal{F}(\mathbf{v}))}{Z}\frac{\partial \mathcal{F}(\mathbf{v}))}{\partial \theta} \\
&= -\frac{\partial \mathcal{F}(\mathbf{v})}{\partial \theta} + \sum_{\mathbf{v}} p(\mathbf{v})\frac{\partial \mathcal{F}(\mathbf{v}))}{\partial \theta} \\
&= -\frac{\partial \mathcal{F}(\mathbf{v})}{\partial \theta} + \sum_{\mathbf{v}} p(\mathbf{v}) \sum_{\mathbf{h}} \frac{p(\mathbf{v},\mathbf{h})}{p(\mathbf{v})}\frac{\partial \mathcal{E}(\mathbf{v},\mathbf{h})}{\partial \theta} \\
&= -\frac{\partial \mathcal{F}(\mathbf{v})}{\partial \theta} + \sum_{\mathbf{v},\mathbf{h}} p(\mathbf{v},\mathbf{h})\frac{\partial \mathcal{E}(\mathbf{v},\mathbf{h})}{\partial \theta} \\
&= -\frac{\partial \mathcal{F}(\mathbf{v})}{\partial \theta} + E_{\mathcal{M}}\left\{\frac{\partial \mathcal{E}(\mathbf{v},\mathbf{h})}{\partial \theta}\right\} \\
&= -E_{\mathcal{D}}\left\{\frac{\partial \mathcal{E}(\mathbf{v},\mathbf{h})}{\partial \theta}\right\} + E_{\mathcal{M}}\left\{\frac{\partial \mathcal{E}(\mathbf{v},\mathbf{h})}{\partial \theta}\right\} \qquad \text{(A.2)}
\end{aligned}
$$