Probabilistic Methods for Network Security

From Analysis to Response

Tammo Krueger

Probabilistic Methods for Network Security

From Analysis to Response

vorgelegt von
Diplom-Informatiker
Tammo Krueger
aus Berlin

von der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
– Dr.-Ing. –

genehmigte Dissertation

Promotionsausschuss:
Vorsitzender: Prof. Dr. Manfred Opper
Gutachter: Prof. Dr. Klaus-Robert Müller
Gutachter: Prof. Dr. Konrad Rieck
Gutachter: Prof. Dr. Michael Meier

Tag der wissenschaftlichen Aussprache: 12. Februar 2013

Berlin 2013
D 83

Acknowledgements

Given my previous odyssey in the oceans of research, I am especially grateful to Prof. Dr. Klaus-Robert Müller for finally providing a home: His constant encouragement and ability to gather both talented and kind people in one spot made my stay at the IDA group an essential part of my life. The support of Prof. Dr. Konrad Rieck was indispensable to major parts of this thesis, be it the discussion of scientific problems, providing data sets to prove the functioning of a method, or just listening to funk music to finish the paper in time. I also want to thank Prof. Dr. Michael Meier who kindly agreed to be a referee of this thesis and to provide his expertise in the domain of intrusion detection and malware analysis.

This work couldn't have been done without the help of many people: First of all, I thank Dr. Mikio Braun for not only giving crucial details for the theoretical preconditions of the fast cross-validation but also for digging up some hidden treasures of the jazz and soul world. Apart from discussing many details of matrix algebra and statistics, Dr. Nicole Krämer also widened my knowledge of contemporary action movies. Christian Gehl provided not only support in data set generation but also during the last months at the Fraunhofer FIRST.

Last but not least, my thanks go to the rest of the IDA group which supported me in many not so obvious ways by chit-chats in the coffee room, always keeping the infrastructure fully functional (including a constant coffee supply and preventing the bureaucracy to overwhelm me), and being such a nice bunch of people.

I acknowledge funding from the German Federal Ministry of Education and Research under the projects ReMIND (01-IS07007A) and ALICE (01IB10003B).

Abstract

Today's computer networks are constantly under attack: apart from a continuous amount of network security threats like denial of service attacks or cross-site request forgeries, new markets like mobile computing and tablets open up new venues for fraudulent monetary exploitation. Thus, cybercrime is a steady driving force for constant malicious innovations which keep on increasing the security gap. While formal methods promise perfect protection, they are often hard to implement and very time-consuming. Hence, they can not keep up with the pace of the modern threat landscape and different, more flexible solutions have to be found.

This thesis shows, how methods from statistics and machine learning can improve the security cycle of analysis, detection and response to threats. By carefully layering probabilistic methods and machine learning techniques we find a design pattern similar to best practice in software engineering: Dividing the overall problem modeling process into physical preprocessing, probabilistic preprocessing and probabilistic modeling we arrive at solid solutions for pressing security problems.

For the *analysis* of network security problems we devise a fully automated procedure for protocol inference and state machine analysis of arbitrary network traces. By first transforming the raw network traffic into session information and embedding the network messages into problem-dependent vector spaces via statistical feature selection tests, we are able to extract complete and correct message templates and abstract state machines of the underlying services leveraging specialized, replicate-aware matrix factorization methods in combination with Markov models. To support the *detection* of network security threats we construct a fast model selection procedure which is capable of choosing the correct learning parameters while saving up to two orders of magnitude of computation time. Deconstructing the overall process into substeps in combination with robust testing procedures leads to a statistically controlled model selection process. We show the applicability of our concepts in the domain of intrusion *response* with an intelligent web application firewall which is able to "heal" malicious HTTP requests by cutting out suspicious tokens and replacing them with harmless counterparts, therefore actively protecting the web server. Open source implementation of major parts of this thesis underline the necessity for freely available solutions to foster future development in the implementation-heavy domain of network computer security.

Computer-Netzwerke sind einer steigenden Bedrohung ausgesetzt, und neue Märkte wie Mobile Computing und Tablet-PCs eröffnen dem Cybercrime immer neue Angriffspunkte. Formale Methoden versprechen zwar beweisbaren Schutz, sind jedoch häufig aufwendig in der Implementierung. Um die Sicherheitslücken zu schließen, werden daher flexiblere und effektivere Methoden benötigt, die mit der Fortentwicklung des Bedrohungspotenzials Schritt halten können.

Ausgehend von den Einzelschritten des Sicherheitsprozesses wird in dieser Arbeit gezeigt, wie Verfahren aus der Wahrscheinlichkeitstheorie und Methoden aus dem maschinellen Lernen effektiv verbunden und eingesetzt werden können, um den Sicherheitsprozess zu unterstützen: Im Bereich der *Analyse* werden durch statistische Tests sinnvolle Repräsentationen für Netzwerkverkehr gefunden, die es ermöglichen, Methoden des maschinellen Lernens zur Extraktion von Mustern anzuwenden. In Kombination mit Markov-Modellen zur Abbildung der abstrakten Statusmaschine des Netzwerkdienstes können somit sowohl die dynamischen als auch die statischen Aspekte von Netzwerkkommunikation probabilistisch erfasst werden. Für die *Detektion* von Angriffen wird eine schnelle Modellselektion präsentiert, die es ermöglicht, die korrekten Parameter für einen Lernalgorithmus sehr viel schneller als mit der herkömmlichen Kreuzvalidierung zu finden. Durch das sequentielle Lernen auf Untermengen der Trainingsdaten wird ein probabilistisches Modell des Verhaltens der einzelnen Parameter erstellt, das mittels robuster statistischer Tests kontinuierlich verkleinert wird, bis eine Konfiguration als signifikanter Sieger feststeht. Im Bereich der *Reaktion* auf Angriffe wird eine intelligente Web-Application Firewall vorgestellt, die mittels spezialisierter Einbettungen für einzelne Teile einer HTTP-Anfrage deren bösartige Teile extrahieren und durch gutartige Inhalte ersetzen kann. Methoden der Anomalieerkennung zusammen mit statistischen Tests führen zu einer vollautomatisierten Konfiguration anhand einer gegebenen Trainingsmenge.

Bei allen Lösungen konnte ein generelles Prinzip ähnlich den Design Patterns im Software-Engineering gewinnbringend zum Einsatz gebracht werden: Durch sorgfältige Zerlegung des Problems in unabhängige Schichten können lokale Probleme effizient gelöst werden. Beginnend mit der physikalischen Vorverarbeitung werden die Ursprungsdaten so aufbereitet, dass sie mit Hilfe der probabilistischen Vorverarbeitung in geeignete, spezialisierte Räume eingebettet werden können. Darauf basierend können im folgenden probabilistischen Modellierungsschritt die Daten mit Methodiken des maschinellen Lernens und der Statistik abgebildet werden, sodass probabilistische Inferenz betrieben werden kann. Open Source Implementierungen der wichtigsten Teile dieser Arbeit unterstreichen die Notwendigkeit frei zugänglicher Lösungen zur effizienten Weiterentwicklung und Forschung im Bereich der Netzwerksicherheit.

Contents

List of Figures

List of Symbols

Vector Space	Appendix A.1, p. 123
$x = (x_1, x_2, \ldots, x_f)^\top$	Vector
$\|x\|$	Norm of x
$d_e(x, y)$	Euclidean distance
$X = (x_{ij})$	Matrix
$\|X\|$	Frobenius norm
Probabilities	Appendix A.2, p. 123
$P(A)$	Probability of event A
$P(A\|B)$	Conditional probability
$X \sim P_X$	Random variable X distributed according to P_X
$E[X]$	Mean of random variable X
$Var[X]$	Variance of random variable X
Markov Models	Appendix A.3, p. 125
$S_1^T := [S_1, S_2, \ldots, S_T]$	Indexed sequence of random variables
$s_1^T := [s_1, s_2, \ldots, s_T]$	Concrete state sequences
P_{S_1}	Initial state probability
$P_{S_t\|S_{t-1}}$	State transition probability
Statistical Tests	Appendix A.4, p. 126
H_0	Null hypothesis
H_1	Alternative hypothesis
α	Type I error (reject H_0, but H_0 is true; significance level)
β	Type II error (accept H_0, but H_1 is true)
$1 - \beta$	Power (reject H_0, and H_1 is true)

More details and definitions can be found in the according appendices.

Chapter 1

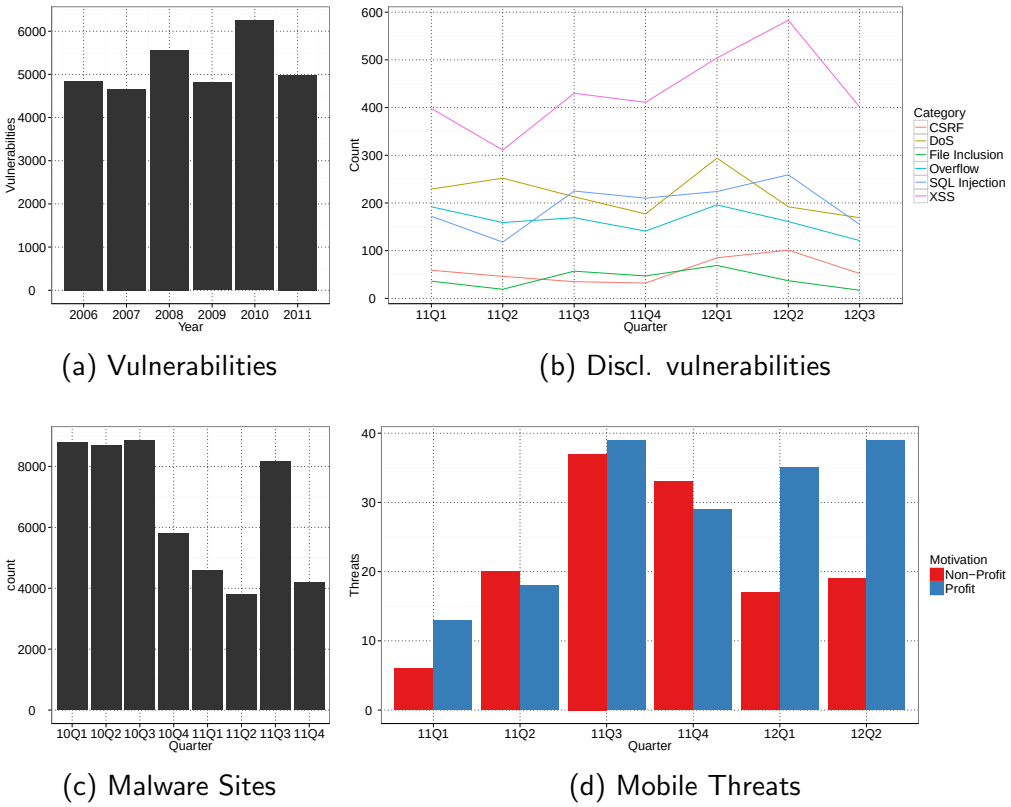
Introduction

1.1 Network Security and Machine Learning

Since the transition of batch-operated systems to time-sharing, multi-user systems, computer security was of paramount interest: Files and information stored by one user should not be readable by other users and actions of one user should not interfere with other users' actions. The need for security was a direct result of the concurrent execution of programs of multiple users on the systems. Gollmann [2002] gives a very pragmatic and concise definition of computer security: "Computer security deals with the prevention and detection of unauthorized actions by users of a computer system." So the interaction of a multitude of users on a system forced the development of safety measures to protect a single user from potential misuse by other users of the system.

First attempts to tackle the problem with formal methods turned out to be extremely difficult: "The difficulties of formal verification, the need for trusted subjects, and the practical infeasibility of entirely eliminating covert channels all indicated, by the end of the 1970s, the technical complexity of computer security."[Mackenzie and Pottinger, 1997]. This effect was even further boosted by the advent of networked systems. As Bishop [2002] states, "with the rise of networking and the Internet, the arena has changed. Workstations and servers, and the networking infrastructure that connects them, now dominate the market. Computer security problems and solutions now focus on a networked environment". Therefore the security research has had to focus even more on the effects and ramifications of the ubiquitous networking environment. So the emergence of networks even complicated matters for the security research community:

> As computing moved beyond the traditional multi-user mainframe, the classical computer security problem had largely been superseded by a host of more diverse problems, many having to do with the integration of computer and communications security in networks; and there was no clear unitary route to the solution of network security. [...] Instead of formal verification of trusted systems progressing downward from formal specifications deeper into systems, as had been anticipated at



(a) Vulnerabilities (b) Discl. vulnerabilities

(c) Malware Sites (d) Mobile Threats

Fig. 1.1: Global threat landscape: The upper left plot shows the number of new vulnerabilities found every year according to Symantec [2012]. The upper right plot shows the number of new vulnerabilities disclosed by type by quarter according to the non-commercial community project OSVDB [2012]. The lower left plot shows the number of new sites actively hosting malware according to MDL [2012] The lower right plot shows the number of mobile threats and their motivation according to F-Secure [2012].

> the start of the 1980s, it remained frozen as "design verification," and even the latter was no more common in practice in the mid-1990s than it had been in the mid-1980s. [Mackenzie and Pottinger, 1997]

Unfortunately, these difficulties have not been solved yet. Constantly evolving technologies even further aggravate the resulting damage. The so-called *security gap* between defenders and defectors [see Schneier, 2012, chapter 16] therefore leads to a rapid increase in potential harm if there are more innovations to exploit. This thesis can be backed up by recent developments in network security: Figure 1.1a shows the number of new vulnerabilities found every year according to Symantec [2012]. Obviously, the innovation of new services and evolution of old ones keep the number of new vulnerabilities, which can potentially be exploited, on a constant high. According to Norton [2012], the costs of cybercrime amount to

$110 billion in just 12 months. Since numbers, especially cost estimates from security related corporations have to be handled with care (see for instance Maass and Rajagopalan [2012] and the references therein for a stimulating discussion), we also validate these numbers from a publicly available information source. The number of new vulnerabilities disclosed by type by quarter according to the non-commercial community project OSVDB [2012] are shown in Figure 1.1b: While not as high as the numbers from Symantec [2012], we can still see that there is a constant source of new vulnerabilities for the last few years. The next plot shows the number of new sites actively hosting malware according to MDL [2012], giving us a clear indication that these vulnerabilities are also actively exploited. New emerging markets like mobile communication and tablets even boost the potential damage: As can be seen in Figure 1.1d, the number of new threats against mobile devices which are profit-oriented is definitely on the rise. Large scale, empirical studies like Bilge and Dumitras [2012] analyzing the prevalence and time windows of zero-day attacks or Dainotti et al. [2012] describing the emergence of a complete Internet scan for vulnerable SIP servers even underline the constant threat potential in today's networked society. In summary, we can see that there is a steady level of both new vulnerabilities and deployment sites for malware. New emerging markets like mobile devices and tablets show an increase in profit-oriented threats displaying that there is an inherently organized driving force which will maintain a constant threat level as long as there is technological development. If we want to stop the expansion of this security gap, additional techniques are needed to confine the damages.

Since formal verification methods of systems have proven to be extremely hard to implement and maintain, other tools from classical computer science and statistics are applied in computer security. Especially solutions bridging the gap between the need of formal rigorousness and practical applicability like machine learning and probabilistic methods in general are promising candidates to solve the problems of computer security. The ability to extract knowledge and rules solely by describing a pool of data by probabilistic models or geometric concepts enables the practitioner to automatically *learn* the information needed from a large enough data base. A first example of such an application of statistical solutions to the problem of intrusion detection showed up in the seminal work of Denning [1987]: By applying probabilistic models like Chebyshev's inequality, Markov process models, or other time series models to audit records, abnormal behavior according to these models can be expressed based on statistical measures. This so-called anomaly based approach to intrusion detection can be seen as a starting point for the application of probabilistic methods and machine learning to computer security.

Years of research generated progress but from the viewpoint of the security experts no conclusive picture: "Similar ideas are in some intrusion detection products, and it is still unclear whether they do a better job than methodically looking for bit patterns that signify an attack. Still, this could someday be a big deal: If fundamental advances ever occur in the field of A[rtificial]I[ntelligence] (a big 'if'), it has the potential to revolutionize computer security" [see Schneier, 2004, page 362]. Similar opinions can be heard in the security community: Sommer and Paxson [2010] criticize the blind application of machine learning methods to security

problems. By presenting new methods, which increase the classification accuracy on some data sets without giving insights to the application domain expert, why and how the method came to its decisions, nothing is gained:

> In particular, we argue for the importance of obtaining *insight* into the operation of an anomaly detection system in terms of its capabilities and limitations *from an operational point of view*. It is crucial to acknowledge that the nature of the domain is such that one can always find schemes that yield marginally better ROC curves than anything else has for a specific given setting. Such results however do not contribute to the progress of the field without any *semantic* understanding of the gain. [Sommer and Paxson, 2010]

Interestingly, in the domain of machine learning a similar, but broader claim was uttered by Wagstaff [2012]: The machine learning community has lost its connection to real-world problems; instead of solving concrete challenges in an application domain and communicating back the results, often evaluations are carried out on some fixed benchmark data sets. As a remedy Wagstaff [2012] proposes some *machine learning impact challenges*, of which one is "a 50% reduction in cybersecurity break-ins through ML defenses".

This whole discussion shows, that there is an inherent need for machine learning and statistics in the field of network and computer security. Classic approaches have failed to keep track with continuous development and security risks are still on the rise. Yet, it is of utter need not to apply these methods blindly; a meaningful solution with an impact in the domain and community should be synchronized with the inner workings of network security and give the domain expert insights into the problems and learned models at hand.

1.2 Probabilistic Methods for Network Security

Successful application of machine learning and probabilistic methods in general require that we understand the needs and demands of the domain experts. In case of computer security, Schneier [2004, page 395] gives a nice overall picture of the domain: "Security is not a product; it's a process. [...] It is vital to understand the real threats to a system, design a security policy commensurate with those threats, and build in appropriate security countermeasures from the beginning."

This basic *security cycle* is depicted in Figure 1.2. This cycle can be read in two directions: For a first time installation we start from the *analysis* of the system to build a *detection* model which implements our security policy. Based on this model we are then able to *respond* to detected threats, therefore delivering appropriate security countermeasures for the system at hand.

For an already implemented system based on these three building blocks, we can read the cycle also in a different direction [Rieck, 2011]: After the *detection* of novel threats, we have to *analyze* them and adjust our prevention system to trigger the correct *response*.

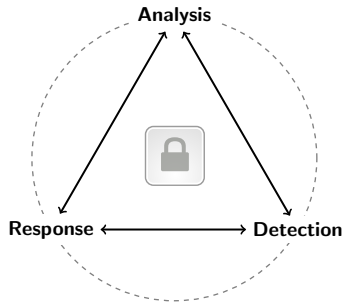


Fig. 1.2: Security cycle.

Obviously, these three building blocks give us a good starting point for a solid and helpful application of machine learning and probabilistic methods to network security: In the domain of *analysis* we can aid the network expert with tools which enable him to get an understanding of the infrastructure that he wants to protect. We see in Chapter 2, that the proposed methods can be used in both directions of the *security cycle*: by analyzing the dynamical aspects of collected network traffic we can build a model of the network traffic exchanged in the infrastructure which helps us to install the right detection model. On the other hand, given a pool of malicious traffic, we can both describe its execution model and give structural features for the accurate implementation of preventive countermeasures.

To learn a *detection* model from data, we first have to choose an appropriate model class and then the correct parameters which are most suitable for the data at hand. For instance, we might decide to solve a certain classification task with a linear ν-support vector machine. Then, we have to find a matching ν parameter which gives an upper bound on the fraction of training errors. These kinds of so-called meta parameters are often tuned via cross-validation, which evaluates a grid of possible parameter combinations by learning a model on part of the training data and estimates the test error on the remaining data. This parameter-tuning process can be quite time-consuming, since we have to learn several models on potentially very large data sets. Chapter 3 shows the utilization of testing procedures to speed up these computations by iteratively learning models on increasing subsets of the data and dropping underperforming models on the way. By using testing procedures we can control the risk of prematurely dropping the right parameter combination and at the same time exploit the information of the whole data set without wasting time for the calculation of unnecessary models.

After selecting a suitable detection model to protect a given network infrastructure, one has to think about finding a *response* mechanism which in case of a detection helps to keep up the protection without harming the functionality of the infrastructure. In Chapter 4 we show how this can be accomplished in the domain of web applications: By setting up an anomaly based web application firewall which automatically learns anomaly models of specific parts of the HTTP protocol from a collection of traffic, we are able to control the incoming requests to a specific web service. If we find an anomalous token in a part of the request we can try to disarm this potentially malicious attempt without risking too much harm, in case the request is just a somehow different, legitimate request.

Throughout this work we will see, that the combination of machine learning methods with the tools of classical statistics often yields a solution which leverages the excellent performance of machine learning with statistical underpinnings a domain expert needs to assess the risk of an operation or to fully understand the details of the data at hand. A layered approach similar to best practice in software design leads to a tool-chain of small, yet powerful steps. Due to decoupling and modularization of each step we achieve an easily debuggable but in its totality

extremely capable system. Each outcome of one step in this tool-chain is the input for the following one. Since each step solves a small part of the whole problem, the inner workings of the methods are still apparent and not hidden inside an obscure algorithm. Similar to the structure of the Unix operating system [Raymond, 2003] this layering approach serves well in all parts of the security cycle: Careful intertwining with statistical procedures in the end leads us to powerful probabilistic methods which helps to solve real-world problems in the domain of network security.

1.3 List of Published Work

During the course of this work several results have been published. In this section we give a brief summary of the main contributions of each chapter and how they are reflected in the published work. Some of the publications form the core of this written thesis and are therefore heavily cited. Furthermore, we introduce the software packages which have been developed and released for public use.

Analysis In the analysis chapter we develop a method to efficiently analyze huge collections of network traces. By embedding the data in a vector space and pruning irrelevant dimensions with statistical testing methods, we can exploit redundancy in replicate-aware matrix factorization methods to find a suitable subspace of even millions of messages very efficiently. After learning the abstract state machine of a service with Markov models and applying automaton minimization algorithms from theoretical computer science, we can extract even finer-grained templates and rules which can ultimately be used to simulate network services. The core of the analysis chapter has been published in Krueger et al. [2011] and Krueger et al. [2012a]. We have released a CRAN (see R Core Team [2012]) package `PRISMA` which is publicly available and contains the replicate-aware matrix factorizations and statistical feature selection methods. Montavon et al. [2012] deal with the general analysis of kernel methods to gain a better understanding of both the learned model and the data problem.

Detection In the detection chapter we develop an efficient model selection procedure based on the representation of each possible model by the expected risk on increasing subsets of the data. By dropping significantly underperforming models on the way we can save substantial computation time with nearly no impact on the accuracy of the selected model. The main part of the detection chapter has been published in Krueger et al. [2011b] and Krueger et al. [2012b] and is currently under review in Krueger et al. [2012c]. We have released a CRAN (see R Core Team [2012]) package `CVST` which is publicly available and contains the complete model selection procedure and all the learners used in the chapter as prepackaged methods suitable for the model selection. Additional publications dealing with the matter of detection in computer security are Rieck et al. [2010], Rieck et al. [2010a], Krueger and Rieck [2012], and Schwenk et al. [2012].

Response In the response chapter we show, how the methods developed in the analysis part can be put to use for intelligent responses in a network intrusion prevention system. By parsing the incoming requests, finding localized embeddings, and learning token-based anomaly models, we can exploit the focused structure of this approach and apply "minimally invasive surgery" to potentially malicious parts of the request. The core of the response chapter can be found in Krueger et al. [2010]. Additional response mechanisms are described in Krueger et al. [2008] with a follow-up study of a real-world deployment of the system in Schuster et al. [2010].

Overall, we will see that a successful application of probabilistic methods require a layered approach similar to good practice in software design: The first step is the physical preprocessing of the data which lays the ground for the probabilistic preprocessing of the problem domain. This allows for a probabilistic description of the data which subsequently is incorporated into powerful probabilistic models. With these models, statistically sound reasoning about the problem domain can be achieved giving the domain experts quantifiable and controllable risks in the application of the found solutions. We have added the core publications in a separate bibliography directly following this section for easier consultation.

T. Krueger and K. Rieck. Intelligent defense against malicious javascript code. *PIK - Praxis der Informationsverarbeitung und Kommunikation*, 35(1):54–60, 2012.

T. Krueger, C. Gehl, K. Rieck, and P. Laskov. An architecture for inline anomaly detection. In *Proceedings of the European Conference on Computer Network Defense (EC2ND)*, pages 11–18, 2008.

T. Krueger, C. Gehl, K. Rieck, and P. Laskov. TokDoc: A self-healing web application firewall. In *Proceedings of the 25th ACM Symposium on Applied Computing (SAC)*, pages 1846–1853, March 2010.

T. Krueger, N. Krämer, and K. Rieck. ASAP: automatic semantics-aware analysis of network payloads. *Proceedings of the ECML/PKDD Conference on Privacy and Security Issues in Data Mining and Machine Learning*, 2011a.

T. Krueger, D. Panknin, and M. Braun. Fast cross-validation via sequential analysis. In *Neural Information Processing Systems (NIPS), Big Learning Workshop*, 2011b. URL `http://biglearn.org/index.php/Papers#paper2`.

T. Krueger, H. Gascon, N. Krämer, and K. Rieck. Learning stateful models for network honeypots. In *Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence*, AISEC '12, 2012a.

T. Krueger, D. Panknin, and M. Braun. Fast cross-validation via sequential testing. *Computing Research Repository*, abs/1206.2248, 2012b. URL `http://arxiv.org/abs/1206.2248`.

T. Krueger, D. Panknin, and M. Braun. Fast cross-validation via sequential testing. *Journal of Machine Learning Research*, in Review, 2012c.

G. Montavon, M. L. Braun, T. Krueger, and K.-R. Müller. Analyzing local structure in kernel-based learning: Explanation, complexity and reliability assessment. *IEEE Signal Processing Magazine*, in review, 2012.

K. Rieck, T. Krueger, U. Brefeld, and K.-R. Müller. Approximate tree kernels. *Journal of Machine Learning Research*, 11:555–580, 2010a.

K. Rieck, T. Krueger, and A. Dewald. Cujo: efficient detection and prevention of drive-by-download attacks. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 31–39, 2010b.

I. Schuster, T. Krueger, C. Gehl, K. Rieck, and P. Laskov. Fips: First intrusion prevention system. Technical Report 1, Fraunhofer FIRST, 2010. URL `http://publica.fraunhofer.de/documents/N-148519.html`.

G. Schwenk, A. Bikadorov, T. Krueger, and K. Rieck. Autonomous learning for detection of javascript attacks: Vision or reality? In *Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence*, AISEC '12, 2012.

Chapter 2

Analysis

Before delving into the subject of analyzing network data, we have to look at the fundamental building block of network communication: The basis for network communication is the Transmission Control Protocol (TCP) and Internet Protocol (IP) suite. Developed in the 1970s, TCP/IP is a solid, layered approach to enable the communication of systems even between different networks. Figure 2.1 shows the steps involved when a client contacts a server on a conceptual level according to RFC 1122 [Braden, 1989]: First, the request has to be encoded according to the application protocol. Then this data is sent down the network stack, where subsequently parts of the transport layer, the internet layer and the physical link layer are added, which control the actual transmission of the data to the corresponding server. This can involve several hops via routers until the packet is received at the server, which strips the packet of the link, internet and transport information and just parses the application data. This parsed request triggers a state change in the underlying execution model of the server and a corresponding reply message can be generated.

At each layer we can start our analysis, for instance by setting up a flow analyzer at a dedicated sensor in the network infrastructure and monitoring the network for uncommon communication patterns on the transport or internet layer [McHugh,

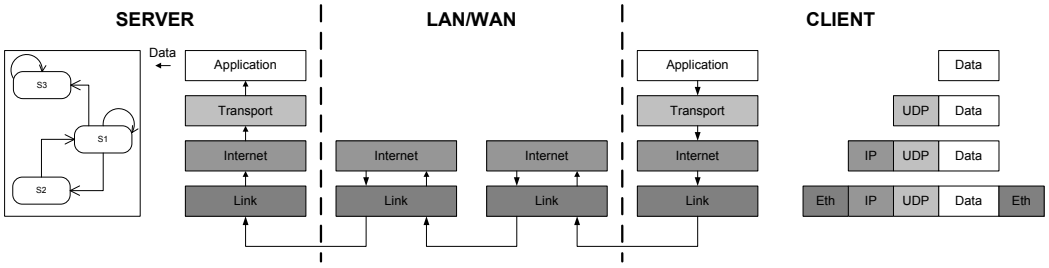
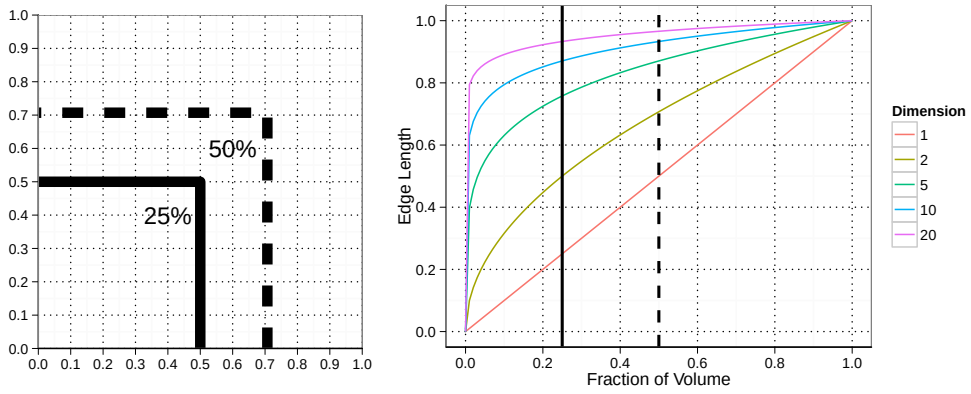


Fig. 2.1: Overview of network communication: The client sends its encoded request down the network stack over the local or wide area network (LAN/WAN). The server retrieves the raw data and is able to decode it and triggers the corresponding reply according to its internal state machine.



(a) Example in two dimensions (b)

Fig. 2.2: Curse of dimensionality: If we want to cover 25% of the volume of the unit square, we need to sample from half the range of each dimension (see left plot). For 50% of the volume this ratio rises to roughly 0.71 and is even bigger for higher dimensions (see right plot).

2004, Taylor et al., 2009]. In this work we focus on the analysis of the application layer, i.e. apart from session information which we infer from the transport layer we ignore the lower layer information and focus on the semantics of the application encoded in the *data* or so-called *payload* exchanged between a client and a server. These *messages* carry the information which kind of action a client wants to perform or what information should be transmitted to accomplish a certain task.

To this end we leverage methods from natural language processing like n-grams and bag-of-words to embed the message in a vector space suitable for further processing. We will see in Section 2.1.2 that this embedding is very high dimensional, which can pose serious problems for learning methods: The so-called *curse of dimensionality* [Bellman, 1961] describes the phenomenon, that the higher the dimension of the input space the bigger the volume of the data space. Therefore, local learning methods suffer from the need to sample from a very wide range of the room to cover the volume for an estimate with a reasonable variance if the data is uniformly distributed in the space. This phenomenon is exemplified in Figure 2.2a for two dimensions: To cover 25% of the volume of the unit square, we have to sample from half of the the range of both dimensions. Similarly, for 50% of the volume we need $\sqrt{0.5} \approx 0.71$ of the total range. Figure 2.2b generalizes this observation for higher dimensions. We can see, that the range we have to sample to cover a specific volume of the space increases rapidly with the dimension.

So if we deal with a high-dimensional space we have to take precautions to make learning in this space feasible. We will see in the course of the treatment, that we can eliminate the bulk of the dimensions by statistical tests, which focus the analysis to those dimensions of the data, which carry interesting information. This *feature selection* is backed up by data compression methods based on matrix

factorization, which exploits structural domain knowledge like part-whole relationships and special tailored distance metrics to model the underlying data in a much smaller subspace which circumvents the problems of these high-dimensional spaces.

While this approach alone can extract valuable information from the data, we can even do better by incorporating dynamical data as encoded in the session information in our analysis: In the workflow of a network service specific kinds of messages are often associated with specific states of the communication. For instance for the file transfer protocol (FTP) a client first has to supply credentials like username and password via the `USER` and `PASS` directive, before he can download files from this server. This session handling is encoded in a state machine which keeps track of the state of each session handled by the server.

Once a system is installed and in active use, specific usage patterns will occur. Apart from workflows built into the protocol itself like the user credential process mentioned above for FTP, other patterns might emerge, which are solely dependent on the task at this specific site. For instance, a web server can host several different web services, which all can implement different workflows on top of the hypertext transfer protocol (HTTP). Thus, it is not sufficient to know the application protocol syntax (in this case the HTTP grammar) but rather to look deeper at the content of the messages to successfully recover the underlying semantics of the workflows. Apart from being a valuable tool for the system administrator at a site, the inferred model of communication can be used for the active protection of this concrete infrastructure: If used in a generative fashion, the model can act as a so-called *honeypot* in the infrastructure. The only purpose of these honeypots is to lure potential attackers and gather information about their intents and the general threat potential in the infrastructure. Furthermore, the extracted message templates give valuable information of the structure and content of the communication that takes place at the site, enabling the system administrator to find focused detection models specially tailored for the site.

In this chapter we present a probabilistic approach to model both the message content and the underlying state machine from network traffic. By inspecting the message interplay between client and server based on a preceding event identification, our model is capable of learning an approximation of the state machine of the service, which not only captures the behavior but can also be used in a generative fashion for simulating long communication sessions.

The main contributions of this *protocol inspection and state machine analysis (PRISMA)* are as follows:

1. Our method is able to learn a stateful model from the network traffic of a service that can be used for simulating valid communication.

2. To construct this model, our method infers the message format, the state machine as well as rules for propagating information between states using probabilistic methods.

3. Our method builds on special tailored embedding and clustering techniques that allow for large-scale applications with millions of network messages.

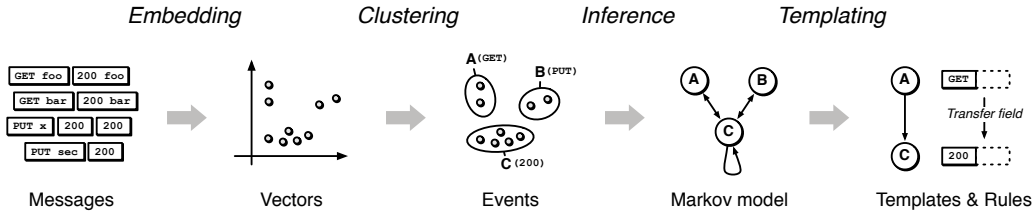


Fig. 2.3: Overview of the Protocol Inspection and State Machine Analysis (PRISMA).

Section 2.1 describes the individual steps of PRISMA. After evaluating PRISMA on different data sets in Section 2.2 and Section 2.3, we give a detailed account of practical extensions to the PRISMA framework and related work in Section 2.4. Further directions and application domains are outlined in Section 2.5, which concludes the chapter. Appendix B.1 gives a complete example of a PRISMA model based on a simple toy problem: Given network traces of a robot communicating with its environment, a behavioral model is learned via PRISMA.

2.1 The PRISMA Method

Given a collection of recorded traffic of a specific network service, the goal of PRISMA is to extract a state machine with associated templates and rules, which describe the information flow from message to message. After a first preprocessing stage, where the raw network traffic is converted to sessions containing messages, our method proceeds in the following steps (see Figure 2.3):

1. To find common structures in the data, we first define a similarity measure between messages. This is done by *embedding* the messages in special vector spaces which are reduced via statistical tests to focus on discriminative features (see Section 2.1.2).

2. We proceed by modeling each session of messages as a sequence of *events*. By leveraging the embedding of the previous step we apply *part-based* or *position-based clustering* which groups individual messages into events (see Section 2.1.3).

3. Each of the extracted sequences of events can be seen as a path through the protocol's state machine. To infer an approximation of this state machine, we use the probabilistic concept of *Markov models*, where transitions are linked with probabilities (see Section 2.1.4).

4. Finally, we automatically generate *templates* for the messages associated with each state of the Markov model and derive *rules* that describe the information flow between the different states during a communication (see Section 2.1.5).

Throughout the chapter we use the term *message* as an atomic exchange of a byte sequence between a client and server. An *event* describes a certain action

on the client or server side which is directly connected with the state machine of the modeled network service. A *template* is a message structure consisting of a sequence of tokens and fields. *Rules* describe the message flow between the fields of consecutive templates instantiated for a concrete session.

2.1.1 Preprocessing of Network Data

To learn the inner structure and behavior of a specific network service we first have to collect sufficient data for the inference. Normally, this can be done at a dedicated sensor, which collects the raw packet data in a binary format for instance via the tool `tcpdump`. Apart from the payload, each packet contains a source and destination address. To actually reconstruct the information flow between a client and a server, these packets have to be re-assembled to eliminate artifacts from the network and transport layer. For this task we have devised a network recorder which uses the mature library `Libnids` for re-assembling TCP and UDP communication streams.

These streams are the input for a session extractor, which generates for each re-assembled packet payload a specific session identifier according to the source and destination of the packet. If two packets occur with a very small delay of τ_{msg} milliseconds, the payloads will be merged. If a specific session identified by its source and destination does not have any more communication within τ_{session} milliseconds, the corresponding session will be flagged as terminated, such that any other message arriving with this specific source/destination combination will open a new session.

This network recorder and session extractor preprocesses the raw network traces into sessions containing messages. In the following we will use this preprocessed data in the subsequent steps of the analysis.

2.1.2 Embedding of Messages

After the preprocessing, a message x can be modeled as sequence of bytes, that is, $x \in B^\star$, with $B = \{0, \ldots, 255\}$. To infer common structures from a pool of messages we need a similarity measure which is capable of focusing the analysis on discriminative features. To account for different styles like binary versus textual protocols we introduce two different embeddings both of which can be compressed via statistical tests to enable a more focused analysis.

Embedding with n-grams

One common approach from the domain of natural language processing is to map byte sequences into a finite-dimensional feature space whose dimensions are associated with *n-grams*, substrings of fixed length n. Formally, we can describe these substrings as $W = B^n$ and define an embedding function $\phi : B^\star \mapsto \mathbb{R}^{|W|}$ as follows

$$\phi(x) := (\phi_w(x))_{w \in W} \quad \text{with} \quad \phi_w(x) := \text{occ}_w(x)$$

which simply records, whether a specific n-gram w occurs in a given string. For instance,

$$\phi(\text{"Hello"}) = (0, \ldots, \overset{\text{Hel ell llo}}{1}, 1, 1, \ldots, 0)^T \in \mathbb{R}^{16777216}$$

for $n = 3$. From this example we can see that the corresponding feature space has a finite but high dimensionality. However, this space is generally sparsely populated, which allows for efficient data representation [Rieck and Laskov, 2008].

Embedding with Tokens

Another well-known concept from the domain of natural language processing is the tokenization of a byte sequence via pre-defined separator characters S. This embedding $\phi : B^\star \mapsto \mathbb{R}^{|W|}$ maps the byte sequence to a feature vector, which records the occurrences of all possible words W according to the separators, that is, $\phi(x) := (\phi_w(x))_{w \in W}$. For example, if we consider the set of separators $S = \{\sqcup\}$, we get the following embedding

$$\phi(\text{"Hey ho, let's go"}) = (0, \ldots, \overset{\text{Hey ho, let's go}}{1}, 1, 1, 1 \ldots, 0)^T \in \mathbb{R}^{|W|}.$$

Similarly to the n-gram embedding, the dimension of the resulting feature space is large but sparsely populated, therefore efficient storage models are also available [Rieck and Laskov, 2008].

Dimensionality Reduction

For finding structure in network communication, the analysis has to focus on features which discriminate the messages in the data pool. Volatile features, like randomly generated nonces or cookies, will only occur once and lead to an unnecessary bloated vector space. The same holds true for constant tokens of the protocol, since their occurrence in a message will be almost certain.

Consequently, we use a statistical test-driven dimension reduction, which allows us to split the feature space as follows: $F = F_{constant} \cup F_{variable} \cup F_{volatile}$. To this end, we apply a binomial test (see Appendix A.4) to each feature, whether it is distributed with a frequency of approximately 1 (corresponding to a constant feature) or 0 (a volatile feature, respectively). After application of a multiple testing correction according to Holm [1979] we keep only those features, which are not constant and not volatile given a statistical significance level of $\alpha = 0.05$. To further simplify the feature space, we unite features which exhibit a correlation near to one.

Given these embeddings and the dimension reduction technique, we are now able to define a data-driven feature space for messages, which allows us to introduce geometrical concepts like metrics (see Appendix A.1). This opens up the whole field of machine learning tools to be applied for network communication. In the following we will assume, that of the total number of features $f = |W|$ the dimension reduction methods keeps just $\tilde{f} = |\widetilde{W}|$ features. The embedding function ϕ is adjusted accordingly, i.e. silently returns just the $\tilde{f}$ reduced features.

2.1.3 Clustering for Event Inference

Messages which occur at specific events in the flow of communication often exhibit similar structural features. Thus, to extract event information we can exploit this structural dependency. Inside a vector space we can define a metric to capture our notion of the similarity between two messages. For instance the Euclidean metric

$$d_e(x, y) := \sqrt{\sum_{w \in \widetilde{W}} (\phi_w(x) - \phi_w(y))^2}$$

calculates the distance between two points based on the occurrence of the $|\widetilde{W}|$ words contained in the corpus. Using these metrics clustering algorithms can be applied to extract common structures in a data pool and thereby indirectly recover the underlying event information.

For inferring structure from network protocol traces we suggest two possible clustering techniques: One for protocols, which are assembled of parts and one for monolithic communication, where tokens are weighted according to their absolute position in the message. Obviously, the experimenter is free to choose an appropriate clustering technique for the data at hand, but we found these two methods to work best with protocol data of the described kind.

Part-Based Clustering via Matrix Factorization

The mapping of network payloads to a vector space induces a geometry, reflecting characteristics captured by the reduced number of feature $\tilde{f}$. For instance, payloads sharing several substrings appear close to each other, whereas network payloads with different content exhibit larger geometric distances. This vectorial representation of network data enables us to identify semantic components *geometrically* (see Appendix A.1). In particular, we apply the concept of matrix factorization for identifying base directions in the vector space. Given a set of payloads $P = \{p_1, \dots p_N\}$ we first define a data matrix A containing the vectors of P as columns by

$$A := [\phi(p_1), \dots, \phi(p_N)] \in \mathbb{R}^{\tilde{f} \times N}.$$

For determining components, we seek a representation of A that retains most information but describes A in terms of few base directions. This can be achieved in terms of a matrix factorization of A into two matrices $B \in \mathbb{R}^{\tilde{f} \times e}$ and $C \in \mathbb{R}^{e \times N}$ such that $e \ll \tilde{f}$ and

$$A \approx BC = \overbrace{\begin{bmatrix} b_1 & \dots & b_e \end{bmatrix}}^{\text{basis}} \underbrace{\begin{bmatrix} c_1 & \dots & c_N \end{bmatrix}}_{\text{coordinates}}. \tag{2.1}$$

The columns $b_1, \dots, b_e \in \mathbb{R}^{\tilde{f}}$ of B form a new basis for the N payloads, where the dimensions of each base direction b_i are associated with the feature set $\widetilde{W}$. As we show in later experiments, this relation of base directions and the feature set can be exploited to construct templates from a matrix factorization. The columns

$c_1, \ldots, c_N \in \mathbb{R}^e$ of C form a new set of coordinates for the payloads in a low-dimensional space. These coordinates can be used for visualization of the data in a low-dimensional space.

Another interesting interpretation of matrix factorization models in the domain of document clustering according to Smyth is as follows: If the documents are generated according to some underlying latent topics $z_j, j \in \{1, \ldots, e\}$ and each topic generates a specific word with probability $P(w_i|z_j)$ and a specific document is assigned to a topic with probability $P(z_j|d)$, then the probability that a specific word occurs in the document d can be expressed as follows:

$$P(w_i|d) = \sum_{j=1}^{e} \underbrace{P(w_i|z_j)}_{b_{ij}} \overbrace{P(z_j|d)}^{c_{jd}},$$

such that the matrix factorization $A = BC$ can also be interpreted as a probabilistic topic model, if the rows in B and columns in C satisfy the axioms of probability (see Appendix A.2).

In general, matrix factorization methods differ in the constraints imposed on the matrices B and C. In this chapter, we study two standard techniques widely used in the field of statistics and data analysis: Principal components analysis (PCA) [Jolliffe, 1986] and non-negative matrix factorization (NMF) [Lee and Seung, 1999].

Principal Component Analysis (PCA) In PCA, we seek base directions, which are orthogonal and capture as much of the variance inside the data as possible. Formally, the ith direction b_i consecutively maximizes the variance of $A^\top b_i$ under the constraint that all base directions are mutually orthonormal:

$$\begin{aligned} b_i \quad = \quad & \arg\max_{\|b\|=1} \operatorname{var}\left(A^\top b\right) \\ & \text{s.t. } b \perp b_j,\ j < i. \end{aligned}$$

In this setting the matrix factorization (2.1) corresponds to a singular value decomposition of A: The L orthonormal basis vectors in B equal the first L left-singular vectors of A, and the coordinates C correspond to the first right-singular vectors of A, multiplied by their singular values. In PCA, all entries of B are typically non-zero, hence each feature contributes to a basis vector b_i. Note, that there exist kernel-based PCA [Schölkopf et al., 1998] which are more powerful but lack the direct interpretability of the extracted components as needed in this setup.

Non-Negative Matrix Factorization (NMF) NMF describes the data by an approximation of the whole embedding matrix $A \in \mathbb{R}^{\tilde{f},N}$ containing N data points with $\tilde{f}$ reduced features by two strictly positive matrices $B \in \mathbb{R}^{\tilde{f},e}, C \in \mathbb{R}^{e,N}$:

$$\begin{aligned} A \approx BC \quad \text{with } (B, C) \quad = \quad & \arg\min_{B,C} \|A - BC\| \\ & \text{s.t. } b_{ij} \geqslant 0,\ c_{jn} \geqslant 0\,. \end{aligned} \tag{2.2}$$

The inner dimension e of the matrix product BC is chosen, such that $e \ll \tilde{f}$ leads to an even more compact representation. Due to the positivity constraint, the matrix B can be interpreted as a new basis (the *parts* of a message), while the matrix C contains the coordinates in this newly spanned space (the *weights* of the different parts). These coordinates are used to ultimately assign a message to a cluster by finding the position with the maximal weight. As shown by Ding et al. [2008], NMF is equivalent to latent semantic indexing by Hofmann [1999], a special kind of topic model, which renders NMF especially useful in the context of document clustering. For instance, the cover illustration shows the tokens of the three base vectors of an NMF applied to a stemmed and tokenized version of the main chapters of this thesis split up by sections. The three components accurately reflect the contents of the three chapters, which are originally distributed over 15 sections.

There are several methods for solving Equation 2.2 (see for instance Paatero and Tapper [1994], Lee and Seung [1999], Hoyer [2004], Heiler and Schnörr [2006]). Here we stick to a practical implementation as introduced in Albright et al. [2006]: Based on the *alternating least squares* approach we alternately solve the following constraint least square problems given the regularization constants λ_B, λ_C

$$\min_{C} \quad \|A - BC\|^2 + \lambda_C \|C\|^2 \tag{2.3}$$

$$\min_{B} \quad \|A^\top - C^\top B\|^2 + \lambda_B \|B\|^2 \tag{2.4}$$

with the corresponding solutions

$$C = \left(B^\top B + \lambda_B I\right) B^\top A \tag{2.5}$$

$$B = \left(CC^\top + \lambda_C I\right) CA^\top . \tag{2.6}$$

The regularization constants can be treated as a meta parameter of the procedure which we choose by cross-validation. Since both the number of features and the number of samples in the matrix A can get quite large (for instance the FTP data set introduced later contains roughly 1.8 million samples and 90,000 features), direct calculation of Equations (2.5) and (2.6) often is infeasible.

Therefore, we devise a reduced, equivalent problem, taking into account that after the dimension reduction step of Section 2.1.2 we have duplicates in our data matrix $A = [a_1, \ldots, a_N]$, i.e. denote by $\widetilde{A} \in \mathbb{R}^{\tilde{f} \times \tilde{N}}$ the matrix without duplicate columns. For the simplification of Equation (2.3) note, that

$$c_i = \left(B^\top B + \lambda_C I\right) B^\top a_i.$$

Hence, we can replace A by $\widetilde{A}$ in Equation (2.5) to obtain $\widetilde{C}$ and then duplicate the resulting $\tilde{c}_i$ accordingly to retrieve C. For the simplification of Equation (2.4) note, that

$$\|A^\top - C^\top B\|^2 = \|\widetilde{A}^\top - \widetilde{C}^\top B\|_W^2 \tag{2.7}$$

with W the $\widetilde{N} \times \widetilde{N}$ diagonal matrix consisting of the number of duplicates of the corresponding column in $\widetilde{A}$. As shown in Holland [1973], the optimization problem of Equation (2.4) with the right side of Equation (2.7) as new objective can be solved by

$$B \;=\; \left(\widetilde{C}W\widetilde{C}^{\top} + \lambda I\right)\widetilde{C}W\widetilde{A}^{\top}\,.$$

These two simplifications allow us to apply NMF even to large data sets with no reduction in accuracy. In Appendix B.2 we describe this replicate-aware version of the NMF algorithm in detail together with the estimation heuristic for the inner dimension e and the applied initialization scheme.

Position-Based Clustering

While matrix factorization techniques are a good choice for protocols, where a message is constructed out of parts, some protocols show very position-dependent features. Since the clustering step in PRISMA is totally independent from the concrete algorithm used, as long as the procedure assigns a cluster label to each message, the experimenter is not fixed to a matrix factorization method but is free to choose an appropriate clustering tool of his choice. To take position-dependent features into account, we propose a weighted distance measure

$$d_w(x,y) := \sqrt{\sum_{w\in\widetilde{W}} (10^{1-p(w,x)}\phi_w(x) - 10^{1-p(w,y)}\phi_w(y))^2},$$

where $p(w,x)$ returns the position of token w in string x. This distance measure can be used to calculate the distance matrix D, which subsequently forms the input to single linkage hierarchical clustering. Note that we can also restrict the calculation of the distance matrix to the reduced data matrix $\widetilde{A}$. This not only saves computing time but also keeps the size of D in a reasonable range.

2.1.4 Inference of State-Machine

Network communication is driven by an underlying state machine, in which certain events trigger certain responses and switches to proceeding states. Sometimes, these switches are probabilistic by nature (for instance a service is temporarily unavailable) or can be modeled as such (for instance in a login procedure 90% of the attempts are successful).

One possible way to model the state machine of a network service in a probabilistic way is by a *hidden Markov model*: The unobserved states correspond to the internal logical states of the service and the messages sent over the network correspond to emitted symbols. Using the Baum-Welch algorithm [Baum and Eagon, 1967] and enough data of service communication, it would be possible to estimate an underlying hidden Markov model to describe the observed data. However, the Baum-Welch algorithm does not guarantee that the found model has the highest likelihood in a global sense.

Learning the Markov Model

Instead of directly trying to infer the underlying hidden Markov model, we start with a regular Markov model which we will later on simplify to a minimal hidden variant. The whole learning process is therefore deterministic and has no inherent randomization like the initial model matrices in the Baum-Welch algorithm (see Appendix A.3 for details). With this approach we circumvent the problem of finding a potential non-optimal model. This determinism comes at a price: it is a well-known fact that hidden Markov models are more powerful than regular ones [Fraser, 2008]. In summary we trade the potential uncertainty with a decrease in model complexity, therefore regularizing the hypotheses space.

Given the session information of the preprocessing step and the label information for each message of the event clustering, we could directly learn a regular Markov model of event sequences by estimating the initial and transition probabilities by their maximum likelihood estimates. However, in this simple Markov model we would drop the direction of the event (i.e. was an event triggered by the client or the server) and limit the history to one message due to the Markov assumption (i.e. the generation of the next event depends just on the previous one). Especially the last limitation would be too strict for network communication, since we would loose the context in which a message would have been generated.

Convoluting the State Space

To circumvent the limitation of the regular Markov model, we use a convoluted and communication-annotated version of the event sequence of a session as follows:

1. Each event will be annotated to reflect, whether it was generated from the client or the server side.

2. For a horizon of k we convolute the annotated and padded event sequence by sliding a window of size k over it and recording the occurring k-tuples.

As an example, assume we have observed the event sequence $[abcd]$ where the messages were generated alternatingly from the client and server. With a horizon of $k = 2$ we would convolute this event sequence to

$$[(\varnothing, \varnothing), (\varnothing, a_C), (a_C, b_S), (b_S, c_C), (c_C, d_S)].$$

So the new, convoluted event space $\widetilde{E}$ will contain $(2|E|+1)^k$ potential events, with $(\varnothing, \varnothing, \ldots, \varnothing)$ being the starting state. By calculating the transition probabilities in this new convoluted event space $\widetilde{E}$ by their maximum likelihood estimates, we specify a regular Markov model with an annotated event horizon of k.

Minimizing the Markov Model

For client server communication a horizon of at least $k = 2$ is necessary to keep the communication context. For more involved processes an even higher horizon might be necessary, which leads to an exponential growth of possible states. We will see in

the evaluation section, that for real network communication this convoluted state space is often very sparsely populated, yet the resulting networks can be large, making the introspection by a human user difficult.

As a remedy we propose the following minimization algorithm to boil down the size of the Markov model while preserving its overall capabilities:

1. Transform Markov model M into a deterministic finite automaton (DFA) $\widehat{M}$:
   a) Keep transitions which have a probability bigger than zero and their associated states.
   b) At each transition the DFA $\widehat{M}$ accepts the new event of the second state (for example the transition connecting state (a_C, b_S) with state (b_S, c_C) would consume event c_C).
2. Apply the DFA minimization algorithm as introduced in Moore [1956] to get the equivalent DFA $\widetilde{M}$ with the minimal number of states but accepting the same language.
3. As a side effect, the algorithm returns an assignment $A_{\widetilde{E},\widetilde{M}}$ of the original states of the convoluted event space $\widetilde{E}$ to the states of the DFA $\widetilde{M}$.

The resulting DFA $\widetilde{M}$ can be used for the inspection of the underlying state model and can be interpreted as a special hidden Markov model: Instead of observing the convoluted events $\widetilde{E}$ we now observe the states of $\widetilde{M}$ according to the assignment $A_{\widetilde{E},\widetilde{M}}$ found by the minimization algorithm. These meta-states subsume equivalent states, and will therefore lead to the acceptance of the same event sequences as the original model. We will show in the evaluation section, that these simplified models drastically decrease the model size and are therefore good candidates for the analysis of the state machine by a human administrator.

2.1.5 Learning Templates and Rules

Each session can be seen as a sequence of events which trigger specific state switches in the state machine. To learn the general information flow during this process, we generalize the messages associated with a state to a template that consists of fixed and variable parts, which often are filled by contents of previous messages. While methods like matrix factorization allow for the extraction of static templates without taking dynamic information into account, session information can be used to even extract such filling rules. So, exploiting the learned Markov model we are now ready to give a procedure to reliably extract templates and rules for the network service at hand.

Inference of Templates

In the event clustering step, we focused on variable, yet neither constant nor volatile features to identify common patterns in the exchanged messages. While this focus makes sense for the identification of underlying events, it is essential to have all features back for the generation of valid, protocol-conformant messages.

	State A_S	State B_C	State C_S
Session 1	`ftp 3.14`	`USER anon`	`331 User anon ok`
Session 2	`ftp 3.12`	`USER ren`	`331 User ren ok`
	⋮	⋮	⋮
Session n	`ftp 2.0`	`USER liz`	`331 User liz ok`
Template	`ftp` □	`USER` □	`331 User` □ `ok`

Fig. 2.4: Example of template generation for a simplified FTP communication.

An additional aspect for the extraction of generic message templates is the underlying state machine of the analyzed service: it is very likely, that the exchanged messages correlate with the current state of the service. Thus, a valid assumption is to assign the messages of each session to its according state in the previously extracted state machine as shown for an artificial example in Figure 2.4: By looking for recurring tokens in each state, generic templates can be constructed containing fixed passages and variable fields according to the distribution in the learning pool.

In more detail, the template inference procedure is structured as follows:

1. Tokenize each message according to the previously chosen embedding.
2. Assign the message of each session to the state of the inferred Markov model.
3. For each state of the Markov model:
   a) Group all assigned messages with the same number of tokens and process each of these groups.
   b) If all messages in a group contain the same token at a specific position, a fixed token is recorded at the resulting template, otherwise a variable field is saved.

At the end of this procedure we will have templates for each state of the Markov model representing the generic messages that might occur. Note that each state might have several different templates assigned according to the observed length distribution: I.e., we simplify the multiple alignment procedure for the extraction of generic templates by focusing the alignment to messages of the same length.

Inference of Rules

Finding rules for filling specific fields in these templates according to previously seen messages now amounts to a simple, yet powerful combination of the Markov model, extracted templates, and session information. For each possible combination of template occurrences of the horizon length k, i.e., $(t_1, t_2, \dots t_k)$:

1. Find all messages which are assigned to these k templates and occur in a session in this exact order.
2. For each field f in the template t_k:

Rule	Description
Copy	Exact copy of the content of one field to another.
Seq.	Copy of a numerical field incremented by d.
Add	Copy the content of a field and add data d to the front or back.
Part	Copy the front or back part of a field split by separator s.
Data	Fill the field by randomly picking previously seen data d.

Tab. 2.1: Rules which are checked during model building. Parameters like d and s are automatically inferred from the training data.

a) Look for a rule to fill f with field content $\hat{f} \doteq f$ of templates $(t_1, t_2, \ldots t_k)$ in $F\%$ of the sessions.

b) If no rule matches, just record the tokens that occur in the training pool (*Data* rule).

The checked rules are described in Table 2.1. This procedure ensures that information that is found in preceding messages which can systematically reproduce contents in a following message in $F\%$ of the cases will get copied over. For instance in the example shown in Figure 2.4 we can observe that in all cases the field of the template associated with state C can be filled with the field of the previous message. The *Data* rule acts as a fallback solution if no match could be found and as a pump-priming rule for the first messages of a session.

2.1.6 Simulation of Network Communication

The inferred PRISMA model now contains three parts: the actual Markov model, the inferred templates and the rule sets associated with these templates. To use these parts of a PRISMA model for simulation of a communication we devised the Lively Essence Network Sensor (LENS) depicted in Algorithm 1.

In addition to the inferred model parts, this module is initialized with the role (client or server) which should be simulated. Note that the PRISMA model itself is role-agnostic and therefore can be used to simulate both sides of a communication. This allows us to even let the model talk to itself by running two instances of LENS with different roles and passing the messages generated from one instance to the other and vice versa.

2.2 Evaluation of Stateless Communication

After presenting the PRISMA method, we turn to an empirical evaluation of its capabilities in different security applications. First, we focus on the evaluation of stateless communication which features the evaluation of the first part of the processing chain, namely the embedding and matrix factorization step. Apart from giving valuable insights to the inner workings of the first steps of our method we

Algorithm 1 The Lively Essence Network Sensor (LENS)

```
1:  function LENS(markovModel, templates, rules, role)
2:      while communication is active do
3:          Wait for message with timeout t
4:          if message M received then
5:              Find matching template T according to the
                current state
6:              Split the message M according to T into fields
7:              Switch the state to the state associated to T
8:          Randomly choose the state S according to the
            transition probabilities of markovModel
9:          if S is in accordance with role then
10:             Find rule set according to the previous
                k (horizon) templates
11:             Apply rules to fill the new template to form
                the message
12:             Send out message
13:             Set current state to S
```

show, that the extracted components of the matrix factorization can be used to construct stateless templates from the resulting base directions. The inclusion of dynamic information of stateful communication in form of session information enables us in Section 2.3 to further refine these templates according to their occurrences in the lifetime of a session.

For the evaluation of stateless content, we first study the framework on a toy data set, which allows us to establish an understanding of how static templates are inferred from communication and compare the performance of the different matrix factorization methods (Section 2.2.1). We then proceed to real-world applications, where network traces containing malicious communication are analyzed for static templates, such as exploited vulnerabilities and attack sources (Section 2.2.2). Finally, we apply our method in the context of network anomaly detection by limiting the processing of network data to the reduced space giving us a huge performance increase while attaining the accuracy of the full data space (Section 2.2.3).

2.2.1 Matrix Factorization Methods

For our first experiment, we consider an artificial data set of HTTP communication where we have total control over protocol syntax and semantics. We simulate a web application supporting three different types of requests, whose network payloads are depicted in Figure 2.5. The first payload reflects a request for static content, the second payload resembles a search query and the last payload corresponds to an administrative request, in which the action parameter is one of the following `rename`, `move`, `delete` or `show`. All requests are equipped with random parts (the name of the static web page, the search string and the administration parameter) to simulate usual fluctuation of web traffic.

```
GET static/3lpAN6C2.html HTTP/1.1
Host: www.foobar.com
Accept: */*
```
Request for static content

```
GET cgi/search.php?s=Eh0YKj3r3wD2I HTTP/1.1
Host: www.foobar.com
Accept: */*
```
Search query

```
GET cgi/admin.php?action=rename&par=dBJh7hS0r5 HTTP/1.1
Host: www.foobar.com
Accept: */*
```
Administrative request

Fig. 2.5: Example payloads of the artificial data set.

Symbols			
1)	`static`	5)	`rename`
2)	`cgi`	6)	`move`
3)	`(search.php ∧ s)`	7)	`delete`
4)	`(action ∧ admin.php ∧ par)`	8)	`show`

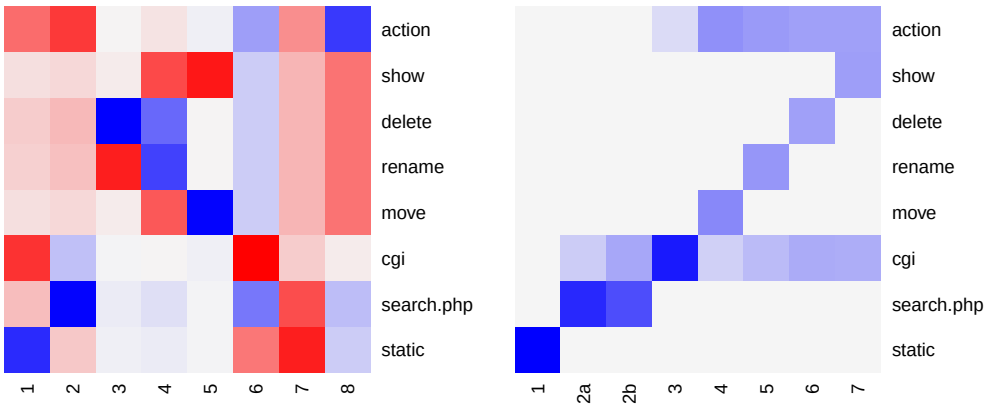
Tab. 2.2: Extracted feature set for the artificial data set. Strings co-occurring in network payloads are coupled using the ∧ operator.

Using this web application, we generate a data set of 1,000 network payloads with a uniform distribution of the three request types. We then apply the first part of the PRISMA method to this data set as detailed in Section 2.1.1 and Section 2.1.2 using tokens as basic strings with delimiters selected according to the specification of HTTP [Fielding et al., 1999]. Based on the extracted feature set, we then apply matrix factorization algorithms, namely Principal Component Analysis (PCA) and Non-negative Matrix Factorization (NMF) for determining base directions in the vector space of payloads. Finally, we construct static templates for these base directions.

We present the extracted feature set in Table 2.2. The feature set consists of eight symbols and corresponds to relevant strings of the underlying web application. Constant and volatile tokens have been filtered, while tokens co-occurring in the communication have been coupled as indicated by the ∧ operator. Note that the feature set does not contain tokens related to HTTP syntax and thereby differs from previous approaches reconstructing protocol grammars and specifications.

Results for the application of matrix factorization algorithms to the artificial data set are visualized in Figure 2.6. For the algorithms PCA and NMF, base directions (matrix B) are shown, where the x-axis details the different directions and the y-axis the contribution of individual feature set symbols.

While both techniques perform a matrix factorization of the payload data, the



(a) PCA (b) NMF

Fig. 2.6: Visualization of bases for PCA and NMF on the artificial data set. Colors signify the intensity of the entry ranging from -1 (red) to 1 (blue).

matrices differ significantly. PCA yields positive and negative contributions in the matrix B indicated by different colors. Although a certain structure and relation of feature set symbols may be deduced from the matrix, a clear separation of different elements is not possible. By contrast, the NMF matrix shows a crisp representation of the base directions. Static and search requests are clearly reflected in individual base directions. The remaining base directions correspond to administrative requests, where different combinations of action types and other feature set symbols have been correctly identified.

Due to this superior performance, we restrict our analysis to base directions determined using the NMF algorithm in the following. Static templates resulting solely from the positive entries of the NMF matrix in Figure 2.6 are presented in Table 2.3. The templates accurately capture the semantics implemented in the simple web application. A set of seven templates is constructed which covers static access of web content, search queries and different administrative tasks. Note that two base directions in Figure 2.6 are identical, resulting in a total of seven static templates. The templates even exhibit hierarchical structure: template 3 resembles a basic administrative request with all following templates being special cases for particular administrative actions.

2.2.2 Analysis of Honeypot Data

Network honeypots have proven to be useful instruments for identification and analysis of novel threats. However, the amount of data collected by honeypots is immense, such that manual inspection of network payloads becomes tedious and futile. The PRISMA method allows for analyzing such big data sets of unknown traffic and extracts semantically interesting network features automatically. We illustrate the utility of our framework on network data collected using the web-based

	Static Templates
1)	`static`
2)	`cgi` ∧ (`search.php` ∧ `s`)
3)	`cgi` ∧ (`action` ∧ `admin.php` ∧ `par`)
4)	`cgi` ∧ (`action` ∧ `admin.php` ∧ `par`) ∧ `move`
5)	`cgi` ∧ (`action` ∧ `admin.php` ∧ `par`) ∧ `rename`
6)	`cgi` ∧ (`action` ∧ `admin.php` ∧ `par`) ∧ `delete`
7)	`cgi` ∧ (`action` ∧ `admin.php` ∧ `par`) ∧ `show`

Tab. 2.3: Static templates extracted for the artificial data set. The templates have been constructed using tokens as basic strings and NMF for matrix factorization.

honeypot *Glastopf*.[1] The honeypot captures attacks against web applications, such as remote file inclusions (RFI) and SQL injection attacks, by exposing typical patterns of vulnerable applications to search engines. The honeypot has been deployed over a period of two months and collected on average 3,400 requests per day. For our experiments, we randomly pick 1,000 requests from the collected data and apply our framework using tokens as underlying feature set. In particular, we extract 40 static templates using the base direction identified by NMF from the embedded HTTP payloads. The templates are shown in Table 2.4.

The extracted static templates can be classified into three categories: *semantics of malware*, *vulnerabilities* and *attack sources*. For example, the first templates reflect different options supplied to a web-based malware. Malicious functionality such as preparing a remote shell (`shellz`), setting up an IRC bouncer (`psybnc`) or scanning for vulnerable hosts (`scannerz`) are clearly manifested in strings of the templates. The following templates characterize vulnerabilities of web applications including corresponding file and parameter names. The following set of templates corresponds to domain and host names used as sources of remote file inclusions. Often not only the originating host but also parts of the complete URL have been discovered. Finally, the method also extracts some miscellaneous templates which cannot directly be mapped to vulnerabilities.

Note that although the templates have been generated from raw HTTP traffic, no syntactic and protocol-specific strings have been extracted, demonstrating the ability of PRISMA to focus on semantics of communication. This feature could aid the security specialist in the overall signature engineering process [see for instance Schmerl et al., 2008] with valuable hints regarding the essential tokens of an attack.

2.2.3 Effectivity of Embedding

To show the effectivity of the embedding we evaluate the capabilities of PRISMA for the application of network intrusion detection. Techniques of anomaly detection are frequently applied as extension to signature-based intrusion detection systems,

[1] Glastopf Web Application Honeypot, see http://glastopf.org

Static Templates	Description
modez ∧ shellz ∧ csp.txt	Semantics of RFI malware
modez ∧ psybnc ∧ csp.txt	—
modez ∧ botz ∧ bot.txt	—
modez ∧ scannerz ∧ bot.txt	—
mosConfig.absolute.path ∧ option ∧ http	Vulnerability (VirtueMart)
mosConfig_absolute_path ∧ option ∧ Itemid	—
com_virtuemart ∧ show_image_in_imgtag.php ...	—
com_virtuemart ∧ export.php ∧ php.txt	—
shop_this_skin_path ∧ skin_shop ∧ standard ...	Vulnerability (Technote)
board_skin_path ∧ Smileys ∧ http	Vulnerability (GNUBoard)
board ∧ skin ∧ http	—
write_update.php ∧ files ∧ 1	—
write_comment_update.php ∧ files ∧ http	—
delete_all.php ∧ admin ∧ zefa.txt	—
delete_comment.php ∧ http ∧ fx29id1.txt	—
appserv ∧ appserv_root ∧ main.php	Vulnerability (Appserv)
_SERVER ∧ DOCUMENT_ROOT ∧ media	Vulnerability (PHP)
error.php ∧ dir ∧ 1	Misc. RFI vulnerabilities
errors.php ∧ error ∧ php.txt ∧ bot.txt	—
administrator ∧ index.php ∧ raw.txt	—
admin ∧ include ∧ http	—
med.buu.ac.th ∧ com_mylink ∧ stealth ...	Sources of attacks
med.buu.ac.th ∧ com_mylink ∧ components ...	—
http ∧ www.hfsb.org ∧ sites ∧ 10225 ∧ img	—
zerozon.co.kr ∧ eeng ∧ zefa.txt	—
http ∧ zerozon.co.kr ∧ photos ∧ count	—
http ∧ musicadelibreria.net ∧ footer	—
qqe.ru ∧ forum ∧ Smileys	—
modules ∧ index.php ∧ files	Miscellaneous templates
data ∧ http ∧ path	—
includes ∧ path ∧ raw.txt	—
%20%20 ∧ http ∧ zefa.txt	—
index.php ∧ option ∧ Itemid	—
http ∧ lib ∧ sourcedir	—
skin ∧ zero_vote ∧ http	—
id2.txt ∧ http ∧ id	—
images ∧ http ∧ bjork.txt	—
spread.txt ∧ media ∧ path	—
bbs ∧ bot.txt ∧ skin	—
id1.txt ∧ http ∧ id	—

Tab. 2.4: Static templates for the honeypot data set. The templates have been constructed using tokens as basic strings and NMF for matrix factorization.

such as the popular `Snort` [Roesch, 1999] and `Bro` [Paxson, 1999] system, since they enable identification of unknown and novel network threats.

For the evaluation of intrusion detection performance, we consider three larger data sets of network payloads: The first data set (FIRST08) contains HTTP requests monitored at the web server of a research institute during a period of 60 days. The second data set (BLOG09) contains requests of several web blogs running the popular platform `WordPress` and spans 33 days of traffic. The third data set (FTP03) comprises the client side requests of FTP sessions recorded over ten days at Lawrence Berkeley National Laboratory [Pang and Paxson, 2003]. Additionally to this benign data, we inject network attacks into the traffic. The attacks are executed in a virtual environment using common tools for penetration testing, such as `Metasploit`, and are carefully adapted to match characteristics of the data sets (see Section 4.2 for details).

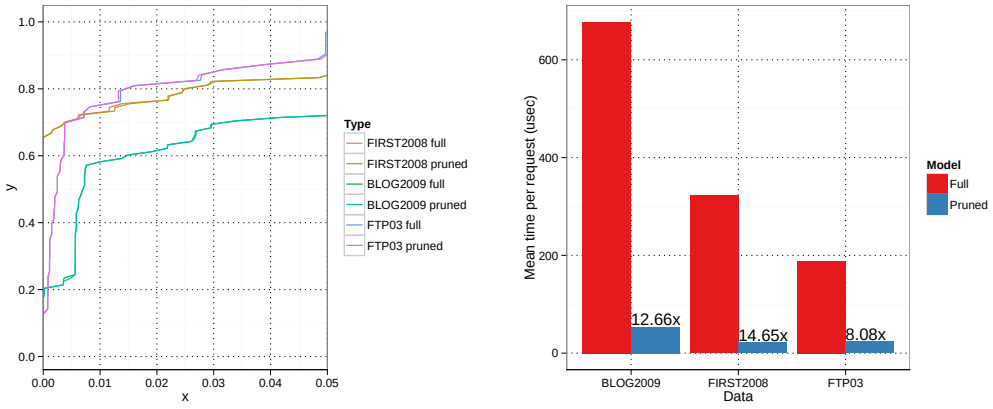
For the experiment, we apply a detection method similar to the work of Wang et al. [Wang and Stolfo, 2004, Wang et al., 2005]. A centroid model of normal network payloads is constructed using n-grams, $\mu_{\text{full}} = \frac{1}{N}\sum_{i=1}^{N}\phi(p_i)$, and used for identifying unusual network content. Additionally, we consider a second model in which the n-grams are refined using the reduced vector space. Formally, after calculating the matrix factorization $A = BC$, we construct this model as follows: $\mu_{\text{reduced}} = B(\frac{1}{N}\sum_{i=1}^{N} c_i)$, where we calculate the centroid in the lower-dimensional space obtained by the first 20 base directions of NMF. The two models are trained on 1,000 randomly drawn payloads for each data set, and anomaly detection is performed on 200,000 randomly chosen HTTP requests and 20,000 FTP sessions.

Results are shown as Receiver Operating Characteristics (ROC) curves in Figure 2.7a. The performance of the full and reduced centroid model is identical on all three data sets. This demonstrates that the base directions identified by NMF capture semantic information of the underlying protocols sufficient for detection of anomalies and attacks. Figure 2.7b details the runtime performance attained by the different models. Reducing the analysis using the lower-dimensional space provides a significant performance gain over regular anomaly detection. Speed-up factors between 8–15 can be observed and clearly indicate the utility of PRISMA as preprocessing step for anomaly detection.

2.3 Evaluation of Stateful Communication

In this section we show, how the PRISMA method benefits from the incorporation of dynamic information. We demonstrate that PRISMA is capable of learning and also simulating network communication from real network traces. To this end we use several network traces recorded via `tcpdump` and plug one part of the data into our processing pipeline and check the quality of the model both according to the remaining data and syntactical and semantical features of the simulated sessions. By this we ensure an evaluation of PRISMA under real-life conditions:

1. Comparison against the held-out sessions assures *completeness* of the models, meaning that the learned models are capable of replaying real sessions as observed in the data pool.



(a) ROC curves (b) Runtime performance

Fig. 2.7: ROC curves and runtime for network anomaly detection. The templates have been constructed using 4-grams as basic strings and NMF for matrix factorization.

2. Checking syntactical and semantical features of the simulated sessions guarantees the *correctness* of the models from a communication perspective.

In Section 2.3.1 we introduce the data sets and discuss the resulting feature spaces after dimension reduction. Then, we look at the general properties of the learned PRISMA models in Section 2.3.2 and the completeness and correctness of these models in Section 2.3.3. We conclude the evaluation with a case study on malware analysis, showing that PRISMA can be useful in application domains beyond honeypots.

2.3.1 Data Sets and Dimension Reduction

For the evaluation of the PRISMA framework we have chosen three representative data sets, of which two are text-based and one is purely binary:

- *SIP:* A data set recorded in a real, medium sized telephony infrastructure containing roughly seven days of communication of 20 participants with different Session Initiation Protocol (SIP) clients.
- *DNS:* Domain Name System (DNS) requests of a home network with seven different clients collected during one day of heavy use.
- *FTP:* File Transfer Protocol (FTP) data set from the Lawrence Berkeley National Laboratory [Pang and Paxson, 2003] containing both client and server requests from ten days of communication.

Naturally, these data sets vary in size: while the SIP data set is a medium-sized pool of roughly 35,000 messages, the DNS data set contains just 6,000 messages. The FTP data comprise of nearly 1.8 million messages rendering it the biggest data

	Size	Dimension	% kept	% unique
SIP	34,958	72,937	0.39%	2.58%
DNS	5,539	6,625	13.15%	35.64%
FTP	1,760,824	87,140	2.17%	0.24%

Tab. 2.5: Properties of data sets: *size* gives the total number of messages in the data set and *dimension* the number of features before the dimension reduction step. *% kept* and *% unique* gives the percentage of features and messages, which are kept after the dimension reduction step.

set of the evaluation. To accommodate the different properties of the data sets, we apply different embeddings: Since SIP and FTP consist of human-readable text, both can be tokenized with the usual whitespace characters. Due to the binary layout of the DNS data, this tokenization approach would not be feasible, therefore we have chosen a 2-gram embedding for DNS. For all data sets we use 90% of the data to learn the PRISMA model and keep the remaining data for the evaluation carried out in Section 2.3.3.

The resulting feature dimensionality reductions and unique messages are shown in Table 2.5. The first thing to note is that once again the dimension reduction step shows excellent behavior: the relative number of kept features ranges from 0.4% for SIP, 13.2% for DNS and 2.2% for the FTP data set showing the extreme focus, which emanates from the dimensionality reduction. A direct consequence of this is the relative number of unique messages for each data set, ranging from 2.6% for SIP, 35.6% for DNS and 0.2% for FTP. The striking difference between DNS and SIP/FTP in terms of reduction can clearly be explained by the different conceptual layouts of the languages: the highly compressed, binary format of the DNS protocol leaves less room for optimization of the feature space, therefore also the number of unique messages after the dimension reduction is higher compared to the other text-based protocols.

Overall, we see that the dimension reduction is highly effective even for binary protocols. By focusing only on the varying parts of the messages and unique messages in this reduced feature space, valuable computation time can be saved and renders the PRISMA approach capable of modeling even big data collections.

2.3.2 Properties of Learned Models

Following the embedding and dimensionality reduction step we apply the event clustering step as described in Section 2.1.3: Both for the SIP and DNS data set we apply the NMF clustering algorithm, since a quick inspection of the data shows, that the part-whole-relationship underlying the NMF algorithm holds for these two data sets. The relative short FTP messages follow a more or less fixed setup, rendering the position-dependent clustering approach better suited for this kind of data.

	# nodes	Coverage	Min. DFA	Coverage
SIP	148	14.5%	100	9.8%
DNS	381	0.8%	153	0.3%
FTP	1,305	0.8%	653	0.4%

Tab. 2.6: Number of nodes for the PRISMA models both for the unoptimized Markov model and the minimal DFA. *Coverage* relates these numbers to the potential number of nodes possible.

	Copy	***Seq.***	***Add***	***Part***	***Data***	**Total**
SIP	1,916	77	135	52	1,793	3,972
DNS	3,142	4	0	0	3,527	6,673
FTP	532	18	253	35	4,671	5,509

Tab. 2.7: Number of different rules of the PRISMA models extracted for the different data sets.

Table 2.6 summarizes the number of nodes of the extracted Markov model for each data set and relates this number to the potential number of nodes which are attainable as described in Section 2.1.4. We see that the total number of nodes for the SIP data set is smallest, yet the relative coverage is highest. For DNS and FTP the absolute number of nodes is higher, but the relative coverage of the potential node space is very sparse, indicating that there is an inherent dependency of relative coverage and estimated number of clusters. Application of the DFA minimization algorithm to the Markov model significantly reduces the number of nodes for the models converting the resulting networks into dimensions manageable by humans.

The corresponding number of rules for each model is shown in Table 2.7. Note that for the n-gram embedding the *Add* and *Part* rules are deactivated, since they are already handled by the *Copy* rule. We see, that all rules are represented. The SIP data set exhibits a higher number of more involved rules compared to all other data sets reflecting the highly redundant structure of this protocol. Both DNS and FTP have an inherent variable part (the server name for DNS and the file names for FTP) which results in a higher number of *Data* rules compared to the SIP data.

Figure 2.8 gives a visual impression of the learned model for the FTP data set. To generate this session we simulated both sides of the communication with our PRISMA model learned on the FTP data set: one model was executed to act as the client and the other one acted as the server. We see in the resulting log, that the session that was generated is valid FTP: Starting with the initial login procedure, the client sets the `TYPE` of the communication to binary data, then enters passive mode and gets a file from the server. Note, that the name of the file from the client request is copied over to the corresponding reply of the server, showing the power of the inferred rules. Obviously, the byte size of 56 is not the proper size of the

```
1  220 <domain> FTP server ( Version wu-2.6.2(1) Mon Dec 30 16:58:35 PST 2001) ready.
2  USER anonymous
3  331 Guest login ok, send your complete e-mail address as password.
4  PASS <password>
5  230 Guest login ok, access restrictions apply.
6  TYPE I
7  200 Type set to I.
8  PASV
9  227 Entering Passive Mode ( 131,243,1,10,9,240 ).
10 RETR groff-perl-1.18.1-4.i386.rpm
11 150 Opening BINARY mode data connection for groff-perl-1.18.1-4.i386.rpm ( 56 bytes).
```

Fig. 2.8: Sample FTP session generated by executing two PRISMA models against each other (one as client, one as server). Data fields are marked by boxes, exact copy rules are filled in gray.

requested file, since it was chosen randomly from the *Data* rule, but the message itself is a valid FTP reply showing the ability of PRISMA to even generate new messages not seen in the training pool before.

2.3.3 Completeness and Correctness

While the previous figures and examples show that the PRISMA method produces relatively condensed models of both the embedding space and the state machine, questions regarding the completeness and correctness of these models are treated in this section.

Completeness

To judge the *completeness* of the models we take the 10% of the held-out data and simulate either the client or the server side to evaluate whether our learned model contains a path, which could generate a session which resembles the data most. Since the transitions in the model are probabilistic, we cannot ensure that the path we choose during the simulation is synchronized with the actual content of the session. For instance, a session might contain a specific branch of the state machine, which occurs just 5% of the time like a server overload error reply or the like. To alleviate this probabilistic effect we repeat each simulation 100 times and introduce a determinism by feeding the first two messages of a session to the model such that the states for the first two messages which are exchanged are aligned.

The results of these simulations are reported in Figure 2.9. We use the normalized Damerau-Levenshtein distance as similarity (1 meaning equality) which counts the number of insertions, deletions, or substitutions necessary to transform one string into another. At each position of a session we take the maximum attained similarity over all repetitions to take account of the probabilistic effect as described before.

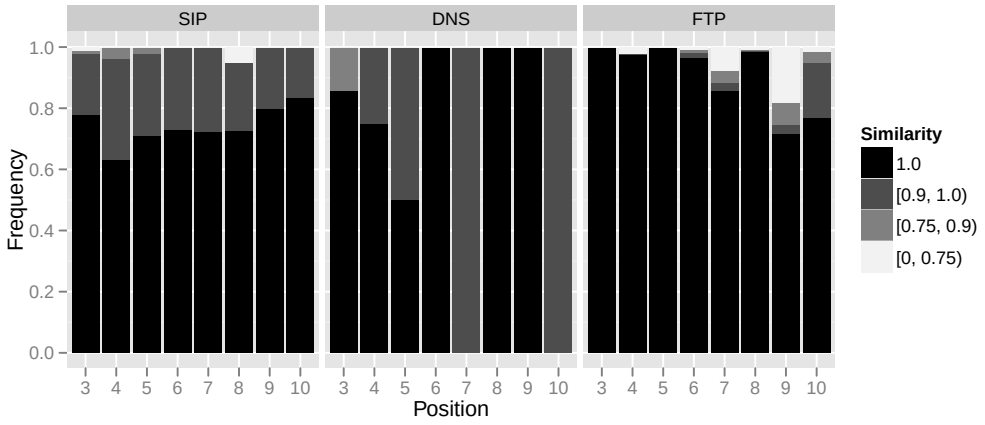


Fig. 2.9: Distribution of maximal similarities by message position in a session replay. Recorded is the normalized edit distance giving a 1.0 for equal messages. The size of the black bar corresponds to the frequency of equal messages, the size of the dark gray bar for similarities ranging between 0.9 and 1.0, the light gray bars for similarities ranging between 0.75 and 0.9.

For the SIP data set we observe that the number of equal messages ranges between 80% and 60%. The similarity score is almost never below 0.9 showing that the learned models can correctly re-model the hold-out session. For DNS this behavior is similar but shows more variance due to the relative low number of sessions having more than six messages. The FTP data set shows an even better performance of the PRISMA model with nearly all messages showing equality up to position six. The frequency of exact resemblance then stays always above 70% showing that even complex protocols can be accurately simulated for more than four steps.

Correctness

Next, we focus on the syntactical and semantical *correctness* of the generated messages. For the syntactical correctness we utilize the protocol filters of the network protocol analyzer `Wireshark`. Only for the FTP protocol we have to check the validity of the commands manually according to the RFCs [Postel and Reynolds, 1985, Hethmon, 2007, Mankins et al., 1980, Hethmon and Elz, 1998]. For the check of semantical correctness we apply the following rules:

- *SIP:* For each message of a session we check whether the `CallID`, `from`- and `to`-tag are preserved, since this triple of values identifies a SIP-session.
- *DNS:* If the message of a session is a reply, we check whether it was queried before in this session and has the same query ID.
- *FTP:* For each FTP request we check, whether both the request and the returned reply code is a valid one according to the RFCs.

	Syntax		Semantic	
	Unidir.	Bidir.	Unidir.	Bidir.
SIP	1.000	1.000	0.988	0.945
DNS	1.000	1.000	1.000	0.994
FTP	0.999	0.821	0.934	0.576

Tab. 2.8: Frequency of sessions having 100% syntactical and semantical correct messages for the different simulation paradigms (uni- and bidirectional).

	Syntax		Semantic	
Msgs. correct	Unidir.	Bidir.	Unidir.	Bidir.
100%	0.999	0.821	0.934	0.576
$> 90\%$	1.000	0.953	0.988	0.878
$> 80\%$	1.000	0.996	1.000	0.982

Tab. 2.9: Breakdown of cumulative syntactical and semantical correctness of sessions for the FTP data.

For each session we count the number of syntactically and semantically correct messages and report the relative frequency of correct messages for the complete session. In addition to the sessions generated for the completeness evaluation (denoted as *unidirectional*) we also simulate 100,000 sessions, in which both sides are generated by our model (denoted as *bidirectional*).

The results are shown in Table 2.8: The syntactical correctness of the sessions is almost always perfect. Only the bidirectional simulations for the FTP data set shows a relative decline of having just 82% percent of the sessions which are totally correct. Regarding the semantics, DNS shows also a nearly perfect behavior. The performance of the SIP model is with 98% and 94% of the sessions totally correct for the uni- and bidirectional simulation, respectively, also in a very good range. While the semantics for the FTP in the unidirectional case show good behavior, the performance declines for the bidirectional simulations: just 57% of the sessions are totally correct. Since FTP sessions tend to be very long, we investigate the correctness in more detail in Table 2.9. By splitting the frequency bins we observe that the bulk of the sessions have more than 80% correct messages. In combination with the higher length of a FTP session this shows that even for difficult, potentially vast communication patterns the PRISMA model is able to capture both the syntax and the semantics of the communication.

In summary, the evaluation shows that the inferred PRISMA models are very compact and show a very high degree of completeness as well as syntactical and semantical correctness. This renders these models ready for the deployment in real-life network infrastructures to act as a honeypot specifically designed for the

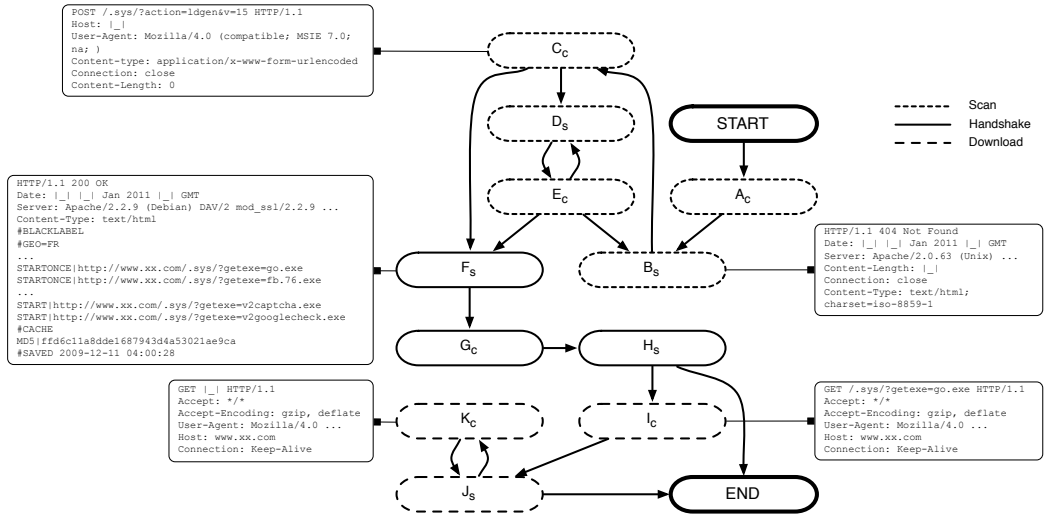


Fig. 2.10: Extracted state model for Koobface traffic: The upper part of the model corresponds to a scanning phase of the malware. The middle part is a handshaking procedure with an infected machine, where the malware gets a new list of malware which is finally downloaded in the lower part of the state machine.

occurring traffic in this network. Contacts to this honeypots can be held up for a high number of steps to gather in-depth information of the behavior and intentions of the potential intruder. This information cannot only be used to estimate the threat potential in an infrastructure at a given time point but also to learn more about the attacks or mischief being conducted.

2.3.4 Case Study: Koobface

In this section we apply PRISMA to network traffic collected from malicious software by Jacob et al. [2011]. We pick one specific class of malware instances and use the token embedding and part-based clustering. We have a total of 147 sessions with 6,674 messages. A detail of the resulting model is depicted in Figure 2.10.

In the upper part we see a scanning loop, in which the malware tries to find an infected server: as long as the server does not answer in a specific format, the scan is continued. After the malware has received a correct reply in state F_S a handshaking procedure between malware and server takes place, which is followed by a download cycle. In state I_C, the malware starts to download the first file from the list (`go.exe`), while in the following states all the other files are downloaded. This can be nicely seen by the *Data* rule associated to the template of state K_C, which contains several instances of the following paths:

```
/.sys/?getexe=tg.14.exe, /.sys/?getexe=ms.26.exe,
/.sys/?getexe=hi.15.exe, /.sys/?getexe=be.18.exe,
/.sys/?getexe=tw.07.exe, /.sys/?getexe=v2captcha.exe,
/.sys/?getexe=v2googlecheck.exe
```

By inspecting the extracted state-machine and the associated templates and rules, a malware analyst can gain insights into the inner workings of a malware instance from the collected network traces alone. This renders PRISMA a valuable tool beyond the realms of honeypot applications, for instance in the construction of multi-step attack signatures [Meier, 2007].

2.4 Discussion and Related Work

In this section we first discuss some valuable extensions for the practical deployment of PRISMA models. While the basic algorithm described in Section 2.1 produce accurate models which reflect the properties of the underlying data pool, the administrator might want to tune these models for a better honeypot performance or to comply to privacy issues. These extensions are discussed in Section 2.4.1. A thorough discussion of related work to the PRISMA framework is given in Section 2.4.2.

2.4.1 Practical Extensions

Deployment of PRISMA models as honeypots adds more practical constraints which are hard to model in an algorithmic fashion. For instance, the model should not contain any information which violates the privacy of any user of the system, since a clever intruder could try to infer this kind of data via the communication with the honeypot. If the data used to learn the model even contains passwords or other security related information the privacy issue even becomes a security risk. Therefore, we have devised some extensions to the PRISMA method which allow an administrator to explore the learned model and adjust it where needed. This part of the analysis framework bridges the gap between the statistical model and the real world: the probabilistic view of the data has to be transformed back into an easily comprehendable and also functioning view for the outside world.

Language Models for Fields

During the rule generation process described in Section 2.1.5 we introduced the *Data* rule as a fallback solution in case we could not find any other rule type associated with this specific field. This rule would trigger the selection of a random value which we have seen during the training phase to fill this specific field. While this is certainly a valid and also sound approach, it is afflicted with some potential privacy problems: If the corresponding field contains privacy or even security related content like usernames or passwords, one would definitely not want to leak this kind of information. Therefore, a more intelligent and pragmatic solution is needed for some fields to preserve privacy or security issues.

One possible solution to this problem is to infer a per-field language model for instance by tokenizing the observed content contained in the *Data* rule by n-grams and model each observed string as a sequence of these n-grams. For a generative model we can then once again leverage the power of Markov chains to model the observed data pool in a probabilistic way. If we now have to fill the field with

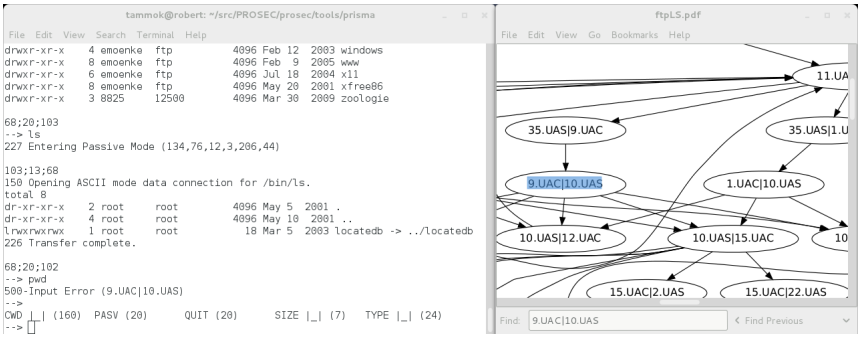


Fig. 2.11: Prototype of a model explorer which allows to "play" with the model and observe its behavior according to the underlying Markov model.

contents, we just have to generate a string according to the learned Markov model. This string will resemble the observed data without directly returning actually observed values. So if the data pool is sufficiently diverse, this kind of model will preserve privacy and security.

Subrules

Another issue with the deployment of a PRISMA honeypot in the real world is to craft a not only correct and complete model of the service, but also an interesting one for potential intruders. It might be that the traffic used for learning the model contains some flaws, which do not allow normally valid input to pass the model. For instance, if we have learned a model with horizon 2 and just observed message sequences of the kind A-B-E-... and C-D-E-..., then the model could not handle the message input A-D-E-... or C-B-E-... since it has not observed such a behavior. If the domain expert knows that such sequences of messages are also valid, an obvious solution to this problem would be just to add the missing edges to the Markov model.

Unfortunately, while the Markov model would now be capable of handling such unseen situations, there would not be any rules in the model which could handle such a case. A not so obvious, yet extremely powerful resort is the concept of subrules: Instead of just learning all rules with horizon k, we could also learn rules for horizon $k-1, k-2, \ldots, 0$ and add them to our model. The LENS module (see Algorithm 1) would automatically pick the rule with the highest possible horizon to carry on the simulation. For instance, in the case above we would have learned the rules for the transition D-E and B-E allowing us to go on with the simulation. Note, that transitions with a horizon of $k = 0$ would consist solely of *Data* rules. This allows the generation of messages even for totally unobserved transitions.

Model Explorer

To extract information like the content of a specific field or how the model reacts to certain inputs, the administrator needs a powerful model explorer. This tool would simulate the communication while showing the underlying model graph.

Each node in this graph could be selected, and associated templates and rules could be inspected and even edited.

Being extremely useful for real-life deployment, programming such a tool with a sophisticated and useable GUI would be a very time-consuming task. Therefore we decided to develop a proof-of-concept model explorer terminal shown in Figure 2.11: The user can play with the model via a command line interface and observe the control flow in a pre-rendered graph of the Markov model. Albeit being a very simple solution, this approach already showed good usability possibilities. Thus, a full fledged model explorer with GUI would definitely be a valuable addition to the PRISMA framework.

Intelligent Template Matching

Simulating network traffic via the LENS module involves finding the right template (see line 5 of Algorithm 1) for a given input. While this can be solved by just tokenizing the given input according to the learned model and find a one-to-one correspondence based on this sequence of tokens in the possible template pool, this approach might be too strict for real-world deployment. Hence, a more intelligent template matching algorithm would be a valuable extension for the LENS module.

One way to approach this problem is by exploiting the edit distance to a given tokenized message to all possible templates: If we do not find a one-to-one match, we can order the possible templates according to their edit-distance to the input and pick the template with the minimal one. Exploiting the information found in the distance matrix of the Damerau-Levenshtein distance for these two strings, matching and omitted tokens could be found resulting in an optimal alignment of the input to the template.

2.4.2 Other Approaches

Inference of both message formats and underlying protocol state machines is not only a relevant problem in today's fast changing networked world but has also been tackled from different directions. From the pure reverse engineering perspective, the open source community has tried to fully understand the inner workings of proprietary protocols in order to develop open implementations of several network services (e.g. SMB, ICQ, Skype). Most of this work has been done in a manual fashion, but the special relevance of network protocol analysis for the security field has led to many research efforts on automatic learning of the protocol state machine and the format of messages involved in a valid communication session.

The work of Beddoe [2005] constitutes a first attempt to extract the fields from protocol messages by drawing upon advanced computational techniques. This approach proposes the clustering of complete messages and the construction of a phylogenetic tree in order to guide the process of global sequence alignment through the Needleman-Wunsch algorithm. With RolePlayer [Cui et al., 2006], the authors build on these ideas to tackle the problem of automatically replaying valid messages from a protocol. Although they present a limited approach that requires the other side of the communication to follow the script that has been

used to configure the system, it already considers the problem of simulating a state dependent communication. Within the same scope is Replayer [Newsome et al., 2006]. The system presented proposes an enhanced solution beyond heuristics, introducing the concepts of theorem proving and weakest pre-condition verification as means to handle protocol dependencies.

A similar approach with a specific security application and also focused on replaying valid messages is introduced in the realm of honeypots by ScriptGen [Leita and Mermoud, 2005, Leita and Dacier, 2006]. This low interaction honeypot learns and simulates communication patterns of vulnerabilities. The objective of ScriptGen is not to infer an accurate specification of the protocol but to obtain the maximum information on the exploitation attempt of a service. Although closely related to our approach, ScriptGen is designed for monitoring low-level attacks against implementations, whereas PRISMA enables collecting and tracking semantic attacks on top of these implementations. In a similar strain of research, Cui and Kannan [2007] have studied the use of tokenization and clustering of individual messages to find fields in message structure. However, this work does not infer the state machine of a protocol and thus can not be used for simulating network communication.

Different approaches based on dynamic taint analysis have been proposed to infer protocol specifications [Caballero et al., 2007, Lin et al., 2008, Wondracek and Comparetti, 2008, Cui et al., 2008]. In order to overcome the lack of semantics of clustering techniques, they rely on dynamic binary analysis of the network service that handles the protocol messages. This eases finding keywords and delimiters but unfortunately all these works defer the task of learning the protocol state machine. An extension to this work with a practical focus on security is carried out by Caballero et al. [2009]. They devise Dispatcher, a system that is capable of infiltrating botnets (whose operation may be based on customized or proprietary protocols), by being able to model, as in our work, messages from both sides of the communication. Also at the host level, it is worth mentioning the work of Wang et al. [2009], which uses binary analysis to extract the properties of messages at memory buffers once they have already been decrypted.

Finally, Comparetti and Wondracek [2009] build on these ideas in order to construct the state machine of a protocol from the dynamic behavior of the application that implements such protocol. The extent of their work certainly resembles ours, nonetheless, our approach is free of the additional burden of binary taint analysis since it is fully network based. The gathering of large amounts of input traces for our system is thereby a straightforward task.

2.5 Outlook and Conclusion

The PRISMA framework proved to be a valuable tool for analyzing network traffic. For stateless communication PRISMA allows the extraction of semantically meaningful components which can be used to describe the essential components of large data collections. The testing-based feature extraction and subsequent embedding in a vector space enables us to apply various tools from machine learning to infer

common structure found in the data pool. Incorporation of state information leads to a tool capable of learning and simulating communication of a given service from network traffic alone. By representing the internal state machine of the service with a Markov model and extracting templates and rules via aligning the collected session to this state machine, PRISMA is able to extract information necessary for an efficient simulation of the data pool. The evaluation shows that both from the viewpoint of completeness and syntactical and semantical correctness PRISMA is capable to emulate real-life network traffic. Application to traces from malware extracts a clear behavioral model of the malware simplifying the daily work of a malware analyst. Apart from the application as an analytical tool PRISMA models can act as a full-fledged abstract state machine and message generator for new, emerging services. This enables a service provider to add a honeypot component to a new service by just learning a PRISMA model from network traffic alone instead of manually programming such a honeypot.

Thus, our next goal is to deploy PRISMA as a honeypot in a dedicated network infrastructure. While our evaluation shows that the PRISMA models are solid we expect valuable input of this real-life application to check our practical extensions and further robustify our approach. Additionally, stateful fuzzing is an interesting other application of PRISMA; for instance, one can use the extracted Markov model to find communication paths inside the state machine, which occur very seldom and therefore should tend to be rather untested and error-prone. We believe that the template structure and the rule can give valuable clues which fields should be fuzzed with what content to maximize the probability of an enforced error. The analysis of Koobface network traces with PRISMA shows that the method can readily be applied in the domain of malware analysis. Still, further refinements, for instance finding the most interesting path in the state machine of the malware, can enhance the usability of PRISMA for this scenario.

In terms of theoretical findings we could show that the course of dimensionality does not interfere with our modeling approach. Obviously, after the preprocessing and feature selection we are able to exploit the fact that the data resides in a much smaller subspace of the high dimensional feature space. Hence, by eliminating unnecessary or redundant features we can expose this subspace even more by using matrix factorization methods like the replicate-aware non-negative matrix factorization. This condensation of the data to its bare, distributional-like minimum allows us in a subsequent step to extract a probabilistic version of the abstract state machine. This layered approach (preprocessing followed by feature reduction followed by probabilistic modeling) is also reflected in the software design of PRISMA where each layer is implemented as a tool chain consisting of small, specialized modules. We have released parts of the PRISMA toolbox as an open source CRAN package (see R Core Team [2012]) named `PRISMA` which contains replicate-aware versions of the NMF and PCA together with import routines and the testing-based feature selection methods for the efficient loading of huge data pools. Hopefully, other users can benefit from the techniques of PRISMA by helping to analyze their data and better protecting their infrastructures.

Chapter 3

Detection

Detecting unwanted behavior in a network infrastructure often involves a rule-based system, which monitors some sensor data and observes aberrations from pre-specified rules. These rules are often hand-crafted from security specialists who analyze their domain and build some meaningful models. For instance we have seen in Table 2.4, that a specific malicious request of a malware has the contents `modez` $\wedge$ `botz` $\wedge$ `bot.txt`. Therefore, the domain expert could build an according regular expression for an intrusion prevention system which constantly monitors network traffic and searches for matches to such regular expression rules in the payloads. Obviously, this solution involves a lot of manpower. While the methods introduced in the previous chapter equip us with powerful tools to support such efforts, it would be even better to *learn* these rules automatically from a given data pool. We have seen in Chapter 2 that the structural features we extracted from the data joint with a similarity metric opens up the whole toolbox of machine learning by transferring the data in a suitable vector space. This allows us to pose the rule learning problem as a regression or classification task.

As a concrete example we give a brief introduction to the Cujo system [Rieck et al., 2010, Krueger and Rieck, 2012]. Cujo acts as a man in the middle between the browser and the web: Whenever a client downloads a page with JavaScript content, the Cujo system inspects the script before delivering it to the client. By looking at both syntactical features encoding the script's structure and dynamical features describing the actual behavior of the script, Cujo decides whether the script is benign and can be delivered to the client or is malicious and should be dropped. The rules to perform these tests are learned *automatically* with a support vector machine (SVM) which takes both malicious and benign scripts as input. By extracting both syntactical and behavioral features with techniques similar to those of Chapter 2 and therefore embedding the data into a vector space, the SVM infers a separating hyperplane with maximal margin [see for instance Müller et al., 2001]. By solving this classification problem we can apply the learned SVM model to new data and decide whether it is a benign or malicious script.

Learning a suitable model for a problem involves the tuning of specific *meta parameters* to find the right fit for the data at hand. In the case of Cujo we have to specify which token embedding should be used: choosing a high-dimensional token

embedding leads to a very fine-grained model which might be too complicated to explain the underlying data generation problem and therefore tends to overfit the data by just memoizing the complete learning pool. On the contrary, choosing a too low-dimensional token embedding might lead to a too simplistic model which underfits the data and does also not capture the underlying data generation process. Thus, finding the right embedding is a crucial task referred to as *model selection*.

In the following we explain the problem of finding the right model in a more formal way for a regression task: In a regression problem we want to learn the prediction equation $y = f(x)$ from a data set consisting of data points $d_1 = (X_1, Y_1), \ldots, d_N = (X_N, Y_N) \in \mathcal{X} \times \mathcal{Y}$ which we assume to be drawn independently and identically distributed from an unknown distribution $P_{X,Y}$. One approach to solve this problem is to minimize the mean squared error (MSE) over a given data set. Basically, without prior knowledge of the underlying data generation process and $P_{X,Y}$, there are infinitely many solutions leading to a minimization of the MSE on the training data. One trivial method would be to just memoize all values from the training data set, which obviously would lead to poor performance on unseen data, i.e., the model would not be able to generalize.

Therefore, one has to impose structural restrictions to the function class and introduce regularizations to the function, which can directly be controlled by so-called *meta parameters*. For instance, in ridge regression we impose a linear function class in combination with a restriction on the resulting coefficient values. As shown by [Hastie et al., 2009, Chapter 7], if we assume that $Y = f(X) + \epsilon$ with $E[\epsilon] = 0$ and $Var[\epsilon] = \sigma_\epsilon^2$, the expected prediction error Err of a regression fit $g(X)$ at an input point x using squared-error loss is:

$$\begin{aligned} Err(X = x) &= E[(Y - g(x))^2 | X = x] \\ &= \sigma_\epsilon^2 + \underbrace{(E[g(x)] - f(x))^2}_{Bias^2[g(x)]} + \underbrace{E[g(x) - E[g(x)]]^2}_{Var[g(x)]} \end{aligned}$$

So the total error can be split up into an irreducible part (variance of the data), the squared bias of the estimate (how far are we from the real solution), and the variance of the estimate (how wiggly is our estimate). The meta parameters of a learning method control the model complexity. The higher the chosen complexity, the less the bias but the higher the variance of the estimate, and vice versa. Thus, if we choose the model complexity too high, we can perfectly imitate the training data (bias tends to zero), but we will suffer from a higher variance on unseen data: the model will overfit. On the contrary, if we choose the complexity too low, our estimate will exhibit a high bias with low variance leading to an underfitted model. Since we are interested in the lowest prediction error, the model selection process has to focus on the optimal point of this bias-variance trade-off at which a further decrease in bias would be dominated by the increase in variance.

Figure 3.1 shows, how the training and test error evolves for ν support vector regression (ν-SVR) models with Gaussian kernels repeatedly trained on 50 training points with varying width of the Gaussian kernel (σ) and fixed regularization parameter (ν): We can observe, that the training error for more complex models (i.e. lower σ) highly underestimates the test error on an independent test set. Fur-

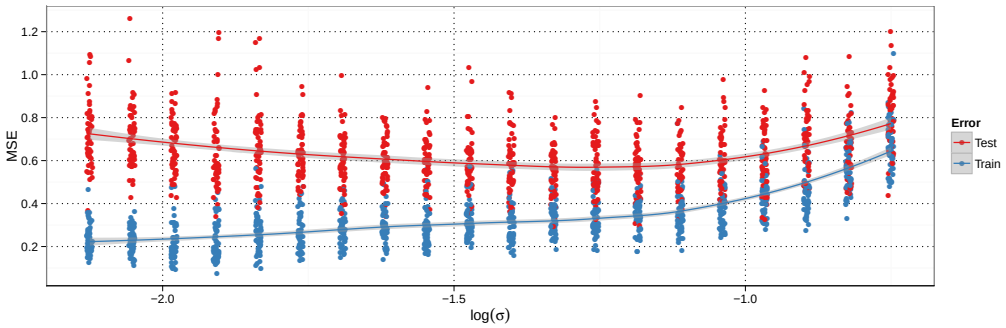
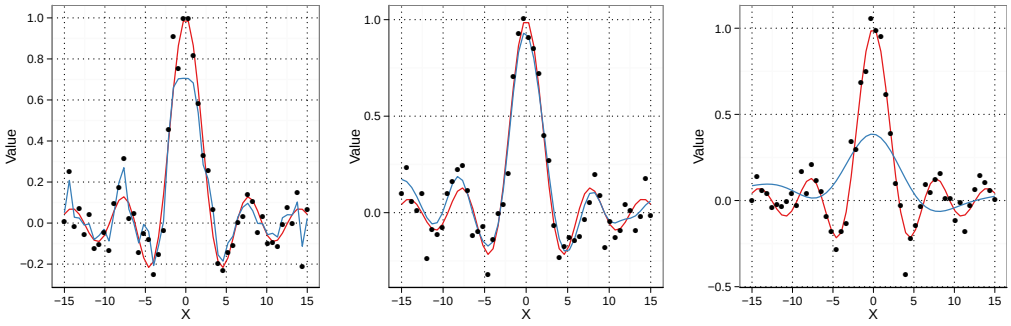


Fig. 3.1: Mean squared error (MSE) of ν support vector regression models with Gaussian kernel repeatedly trained on 50 training points with varying σ. The blue line denotes the MSE on the training set, while the red line denotes the MSE on an independent test set. For high model complexity (i.e. low σ) the training error diverges even more from the real test error due to an overfitting effect.



(a) $\log(\sigma) = -2.25$ (b) $\log(\sigma) = -1.25$ (c) $\log(\sigma) = -0.25$

Fig. 3.2: Prediction (blue line) and real values (red line) of ν support vector regression models with Gaussian kernel trained on 50 training points (black dots) of the *noisy sinc* data set. We see that models with high model complexity (i.e. low σ, left plot) exhibit overfitting while models with very low model complexity (i.e. high σ, right plot) clearly underfit the data.

thermore we can see, that the ν-SVR has an optimal performance roughly around $\log(\sigma) = -1.25$. A visual impression of the different complexity classes is shown in Figure 3.2: while the high complexity model in Figure 3.2a overfits the training data and the low complexity model in Figure 3.2c underfits the real function the optimal model in Figure 3.2b accurately describes the real function class.

Since we cannot rely on the training error of a model to determine the optimal model complexity, we have to find a tool to estimate the test error for the adjustment of the model complexity via the meta parameters of the learning method. Cross-validation is the de-facto standard in applied machine learning to tune meta parameters of machine learning methods in supervised learning settings (see Mosteller and Tukey 1968, Stone 1974, Geisser 1975 and also Arlot et al.

2010 for a recent and extensive review of the method). Part of the data is held back and used as a test set to get a more unbiased estimate of the true generalization error. Cross-validation is computationally quite demanding though. Doing a full grid search on all possible combinations of parameter candidates quickly takes a lot of time, even if one exploits the obvious potential for parallelization. Furthermore, the size of today's data sets often forbids to calculate the performances for all parameters on the complete data. For instance, in network computer security we can often collect huge amounts of data by just tapping the wire at a specific point in the infrastructure and record the traffic. Model selection then also amounts to finding a small enough subset which is sufficient to capture the complexity of the data generation process.

Therefore, cross-validation is seldom executed in full in practice, and different heuristics are usually employed to speed up the computation. For example, instead of using the full grid, local search heuristics may be used to find local minima in the test error (see for instance Kohavi and John 1995, Bengio 2000, Keerthi et al. 2006). However, in general, as with all local search methods, no guarantees can be given as to the quality of the found local minima. Another frequently used heuristic is to perform the cross-validation on a subset of the data, and then train on the full data set to get the most accurate predictions. The problem here is to find the right size of the subset. Apparently, if the subset is too small and cannot reflect the true complexity of the learning problem, the parameters selected by cross-validation will lead to underfitted models. On the other hand, a too large subset will take longer for the cross-validation to finish.

Applying those kinds of heuristics requires both an experienced practitioner and a high familiarity with the data set. However, the effects at play in the subset approach are more manageable: as we will discuss in more depth below, given increasing subsets of the data, the minimizer of the test error will converge, often much earlier than the test error itself. Thus, using subsets in a systematic way opens up a promising way to speed up the model selection process, since training models on smaller subsets of the data is much more time-efficient. During this process care has to be taken when an increase in available data suddenly reveals more structure in the data, leading to a change of the optimal parameter configuration. Still, as we will discuss in more depth, there are ways to guard against such change points, making the heuristic of taking subsets a more promising candidate for an automated procedure.

In this chapter we will propose a method which speeds up cross-validation by considering subsets of increasing size. By removing clearly underperforming parameter configurations on the way this leads to a substantial saving in total computation time as sketched in Figure 3.3. In order to account for possible change points, sequential testing [Wald, 1947] is adapted to control a *safety zone*, roughly speaking, a certain number of allowed failures for a parameter configuration. At the same time this framework gives statistical guarantees for dropping clearly underperforming configurations. Finally, we add a stopping criterion to watch for early convergence of the process to further speed up the computation. Thus, the contributions of this chapter are as follows:

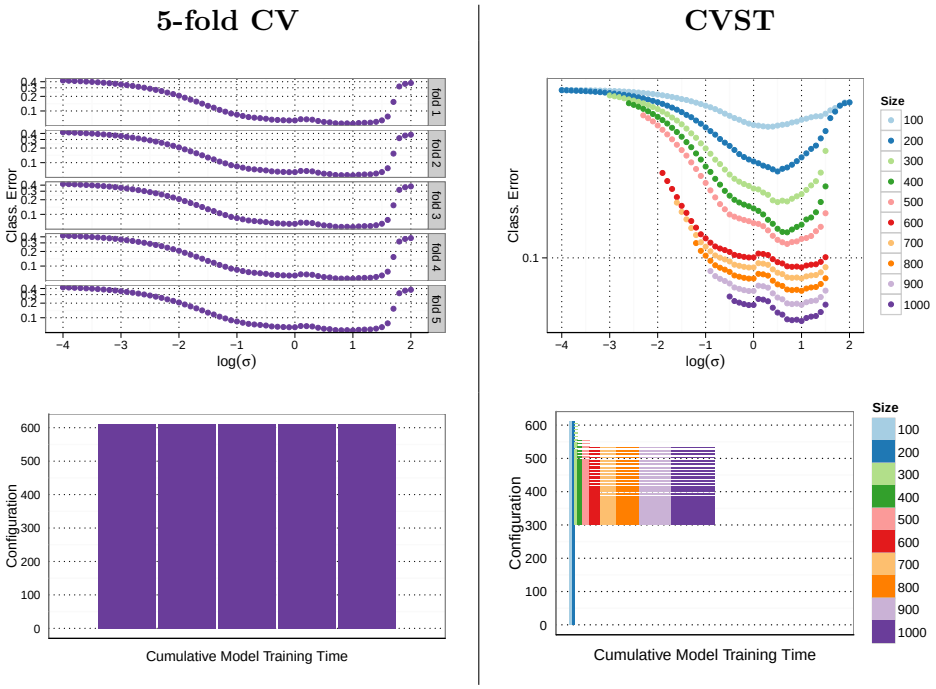


Fig. 3.3: Performance of a 5-fold cross-validation (CV, left) and fast cross-validation via sequential testing (CVST, right): While the CV has to calculate the model for each configuration (here: σ of a Gaussian kernel) on the full data set, the CVST algorithm uses increasing subsets of the data and drops significantly underperforming configurations in each step (upper panels), resulting in a drastic decrease of total calculation time (sum of colored area in lower panels).

1. We describe the overall model performance by the expected risk of a learner on increasing subsets of the data exploiting the probabilistic convergence of this risk on subsets.

2. By transforming these performances in each step via robust and pooled testing procedures into a binary trace matrix we can apply sequential testing procedures which drop statistically significant loser configurations on the way.

3. Theoretical considerations prove the existence of a user controllable safety zone and a resulting error bound of the whole process.

In the following, we will first discuss the effects of taking subsets on learners and cross-validation (Section 3.1), present our method Fast Cross-Validation via Sequential Testing (CVST, Section 3.2), discuss the theoretical properties of the method (Section 3.3) and finally evaluate our method on synthetic and real-world data sets in Section 3.4. Section 3.5 gives an overview of related approaches and Section 3.6 concludes the chapter.

3.1 Cross-Validation on Subsets

Our approach is based on taking subsets of the data to speed up cross-validation. The main question is therefore whether we can reliably estimate the best parameter configuration already from subsets of the data. Under mild conditions we will prove general convergence of the estimate, but also discuss why we can expect the estimate to be much better in practice than the theoretical results developed below.

For the sake of simplicity, we will consider the following slightly simplified formalization of the cross-validation procedure which replaces the empirical test error by the expected risk on a test set, thereby also removing the repetitions involved in a typical run of k-fold cross-validation. That way, we do not have to deal with the additional inaccuracies involved by estimating the test error from a finite set of data.[1]

Assume that our N training data points d_i are given by input/output pairs $d_i = (X_i, Y_i) \in \mathcal{X} \times \mathcal{Y}$ drawn i.i.d. from some probability distribution $P_{X,Y}$ on $\mathcal{X} \times \mathcal{Y}$. As usual, we also assume that we have some loss function $\ell\colon \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ given such that the overall error or expected risk of a predictor $g\colon \mathcal{X} \to \mathcal{Y}$ is given by $R(g) = E[\ell(g(X), Y)]$ where $(X, Y) \sim P_{X,Y}$.

For some set of possible parameter configurations C, let $g_n(c)$ be the predictor learned for parameter $c \in C$ from the first n training examples. Cross-validation basically tries to identify the best parameter c for a given training set size n which minimizes the expected risk. Normally, this is done by computing the best parameter configuration in terms of the empirical risk on a hold-out test set, but for the present discussion, we assume that this estimate is sufficiently accurate for the expected risk.

If we now also consider different training set sizes, we are effectively dealing with the task of minimizing a sequence of functions $e\colon \mathbb{N} \to C \to \mathbb{R}$ defined by

$$e_n(c) = R(g_n(c)). \tag{3.1}$$

We are thus interested in how the minimum at some subset size m of e_m relates to that of e_n. This question is linked to the asymptotic behavior of e_n itself, because if the minimum of e_n converges, so will e_m to e_n eventually.

Now in general, we cannot expect e_n to converge at all. For example, it might be that the model parameter is encoded in a way which needs to be scaled with the sample size, i.e., it might be that a value c at sample size n corresponds to $f(n)c$ for some function $f(n)$, or more complex settings. Under such conditions, a parameter configuration c might work well at a subset size m but become a suboptimal choice for larger sample sizes.

[1] To make this argument more precise: Actually, we would have to compare the error $e_n(c)$ (defined in Equation 3.1, below) against another empirical error $e'_m(c)$ defined on an independent sample. Instead, we compare $e_n(c)$ against $e(c)$, the expected error of parameter configuration c. Technically, this would mean that we have to consider $|e_n(c) - e'_m(c)| \leqslant |e_n(c) - e(c)| + |e'_m(c) - e(c)|$, and we get essentially the same results with an additional limit process of $m \to \infty$, that is, considering the limit as the test set size also tends to infinity. For the sake of simplicity, we have dropped this detail.

We will therefore assume that for each fixed c, $\lim_{n\to\infty} e_n(c)$ exists. Then, using standard techniques, it is straightforward to prove the following result (see Appendix B.3 for the proof).

Theorem 1 *Let C be a finite set and assume that for each fixed c, $e_n(c) \to e(c)$ in probability. Then, the following holds:*

1. ***(Uniform convergence over C)** As $n \to \infty$,*

$$\max_{c\in C} |e_n(c) - e(c)| \to 0$$

   in probability.

2. ***(Convergence of the minimum)** Let c_n^* be such that $e_n(c_n^*) = \min_{c\in C} e_n(c)$, and c^* such that $e(c^*) = \min_{c\in C} e(c)$. Then,*

$$e(c_n^*) - e(c^*) \leqslant 2 \max_{c\in C} |e_n(c) - e(c)|$$

   Moreover,

$$e(c_n^*) - e(c^*) \to 0 \text{ in probability.}$$

3. ***(Evaluating on subsets of the data)** For each ε, $\delta > 0$, there exists a number l such that for all $n \geqslant l$ and m with $l \leqslant m \leqslant n$,*

$$P\{e_n(c_m^*) - e_n(c_n^*) > \varepsilon\} \leqslant \delta.$$

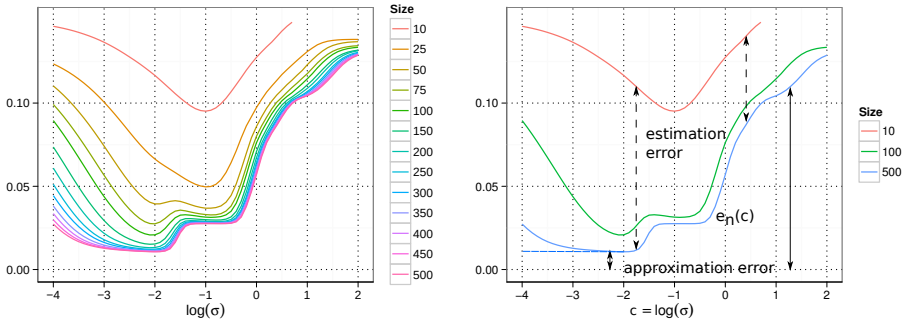
For the sake of simplicity, we have assumed that C is finite. These results can likely be extended to continuous parameter spaces with significant technical overhead.

Theorem 1 proves that, asymptotically, we can expect to get good estimates for the right choice of parameter configurations at training size n on a subset of size m. This result assumes that parameter configurations are encoded in a way which is independent of the size of the training set and hinges on uniform convergence over all possible parameter choices.

Now how well does this result describe the practical findings? Figure 3.4a shows the test errors for a typical example. We train a support vector regression model (SVR) on subsets of the full training set consisting of 500 data points. The data set is the *noisy sinc* data set introduced in Section 3.4.1. Model parameters are the kernel width σ of the Gaussian kernel used, and the regularization parameter, where the values shown are already optimized over the regularization parameter for the sake of simplicity.

We see that the minimum converges rather quickly, first to the plateau of $\log(\sigma) \in [-1.5, -0.3]$ approximately, and then towards the lower one at $[-2.5, -1.7]$, which is also the optimal one at training set size $n = 500$. We see that uniform convergence is not the main driving force. In fact, the errors for small kernel widths are still very far apart even when the minimum is already converged.

In the following, it is helpful to continue the discussion within the empirical risk minimization framework. We assume that the learner is trained by picking



(a) (b)

Fig. 3.4: Test error of an SVR model on the *noisy sinc* data set introduced in Section 3.4.1. We can observe a shift of the optimal σ of the Gaussian kernel to the fine-grained structure of the problem, if we have seen enough data. In Figure (b), approximation error is indicated by the black solid line, and the estimation error by the black dashed line. The asymptotic approximation error is plotted as the blue dashed line. One can see that uniform approximation of the estimation error is not the main driving force, instead, the decay of the approximation error with smaller kernel widths together with an increase of the estimation error at small kernel widths makes sure that the minimum converges quickly.

the model which minimizes the empirical risk over some hypothesis set H. In this setting, one can write the difference between the expected risk of the learned predictor $R(g_n)$ and the Bayes risk R^* as follows (see also Section 12.1 in Devroy et al. [1996] or Section 2.4.3 in Mohri et al. [2012])

$$R(g_n) - R^* = \underbrace{\left(R(g_n) - \inf_{g \in H} R(g)\right)}_{\text{estimation error}} + \underbrace{\left(\inf_{g \in H} R(g) - R^*\right)}_{\text{approximation error}}.$$

The estimation error measures how far the chosen model is from the one which would be asymptotically optimal, while the approximation error measures the difference between the best possible model in the hypothesis class and the true function. Using this decomposition, we can interpret the figure as follows (see Figure 3.4b): The kernel width controls the approximation error. For $\log(\sigma) \geqslant -1.8$, the resulting hypothesis class is too coarse to represent the function under consideration. It becomes smaller until it reaches the level of the Bayes risk as indicated by the dashed blue line. For even larger training set sizes, we can assume that it will stay on this level even for smaller kernel sizes.

The difference between the blue line and the upper lines shows the estimation error. The estimation error has been extensively studied in statistical learning theory and is known to be linked to different notions of complexity like VC-dimension [Vapnik, 1998], fat-shattering dimension [Bartlett et al., 1996], or the norm in the reproducing kernel Hilbert space (RKHS) [Evgeniou and Pontil, 1999]. A typical

result shows that the estimation error can be bounded by terms of the form

$$R(g_n) \leqslant R_n + O\left(\sqrt{\frac{d(H)\log n}{n}}\right),$$

where $d(H)$ is some notion of complexity of the underlying hypothesis class. For our figure, this means that we can expect the estimation error to become larger for smaller kernel widths.

So basically, if we order the parameter configurations according to their complexity, we make three observations:

1. For parameter configurations with small complexity (that is, large kernel width), the approximation error will be high, but the estimation error will be small.

2. For parameter configurations with high complexity, the approximation error will be small, even optimal, but the estimation error will be large.

3. Also, as we see in Figure 3.4b, the approximation error seems to decrease faster with increasing complexity than the estimation error increases.

In combination, the estimates at smaller training set sizes tend to underestimate the true model complexity, but as the approximation error quickly decreases, the minimum also converges to the true one. The fact that the estimation error is larger for more complex models acts as a guard to choose too complex models.

Unfortunately, existing theoretical results are not able to bound the error sufficiently tightly to make these arguments more exact. In particular, the speed of the convergence on the minimum hinges on a tight lower bound on the approximation error, and a realistic upper bound on the estimation error. Approximation errors have been studied for example in the papers by Smale and Zhou [2003] and Steinwart and Scovel [2007], but the papers only prove upper bounds and the rates are also worst-case rates which are likely not close enough to the true errors. On the other hand, the mechanisms which lead to fast convergence of the minimum are plausible when looking at concrete examples as we did above. Therefore, we will assume in the following that the location of the best parameter configuration might initially change but then become more or less stable quickly. We will use sequential testing to introduce a *safety zone* which ensures that our method is robust against these initial changes.

3.2 Fast-Cross Validation via Sequential Testing (CVST)

Recall from Section 3.1 that we have a data set consisting of N data points $d_1 = (X_1, Y_1), \ldots, d_N = (X_N, Y_N) \in \mathcal{X} \times \mathcal{Y}$ which we assume to be drawn i.i.d. from $P_{X,Y}$. We have a learning algorithm which depends on several parameters collected in a configuration $c \in C$. The goal is to select the configuration c^* out of all possible configurations C such that the learned predictor g has the best generalization error with respect to some loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$.

Our approach attempts to speed up the model selection process by learning just on subsamples of size $n := s\frac{N}{S}$ for $1 \leqslant s \leqslant S$, starting with the full set of configurations and eliminating clearly underperforming configurations at each step s based on the performances observed in steps 1 to s. The main loop of Algorithm 2 on page 64 executes the following parts at each step s:

❶ The procedure learns a model on the first n data points for the remaining configurations and stores the test errors on the remaining $N - n$ data points in the pointwise performance matrix P_p (Lines 10-14). This matrix P_p is used on Lines 15-16 to estimate the top performing configurations via robust testing and saves the outcome as a binary "top or flop" scheme accordingly.

❷ The procedure drops significant loser configurations along the way (Lines 17-19) using tests from the sequential analysis framework.

❸ Applying robust, distribution free testing techniques allows for an early stopping of the procedure, when we have seen enough data for a stable parameter estimation (Line 20).

In the following we will discuss each step in the algorithm. A conceptual overview of one iteration of the procedure is depicted in Figure 3.5 for reference.

3.2.1 Robust Transformation of Test Errors

To robustly transform the performance of configurations into the binary information whether it is among the top-performing configurations or turns out to be a flop, we rely on distribution-free tests. The basic idea is to calculate the pointwise performance of a given configuration on data points not used for the learning of the model and find the group of best configurations, which show a similar behavior.

We exemplify this procedure by the situation depicted in Figure 3.5 with K remaining configurations $c_1, c_2, \ldots, c_K$ which are ordered according to their mean performances (i.e. sorted ascending with regard to their expected loss). We now want to find the smallest index $k \leqslant K$, such that the configurations $c_1, c_2, \ldots, c_k$ all show the same behavior on the remaining data points $d_{n+1}, d_{n+2}, \ldots, d_N$ not used in the current model learning process.

The rational behind our comparison procedure is three-fold: First, by ordering the configurations by the mean performances we start with the comparison of the currently best performing configurations first. Second, by using the first $n := s\Delta$ data points for the model building and the remaining $N - n$ data points for the estimation of the average performance of each configuration we compensate the error introduced by learning on smaller subsets of the data by better error estimates on more data points. I.e., for small s we will learn the model on relatively small subsets of the overall available data while we estimate the test error on relatively large portions of the data and vice versa. Third, by applying test procedures directly on the error estimates of individual data points we exploit a further robustifying pooling effect: if we have outliers in the testing data, all models will be affected by this and therefore the overall testing result will not be affected.

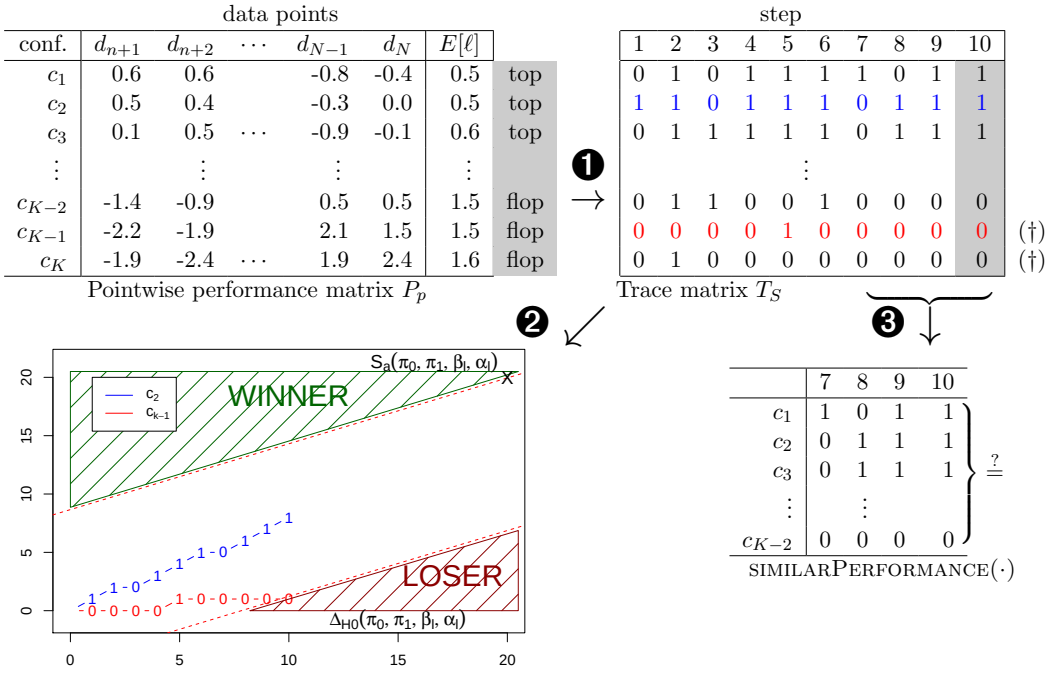


Fig. 3.5: One step of CVST. Shown is the situation in step $s = 10$. ❶ A model based on the first n data points is learned for each configuration (c_1 to c_K). Test errors are calculated on the remaining data (d_{n+1} to d_N) and transformed into a binary performance indicator via robust testing. ❷ Traces of configurations are filtered via sequential analysis (c_{K-1} and c_K are dropped). ❸ The procedure checks whether the remaining configurations perform equally well in the past and stops if this is the case. See Appendix B.7 for a complete example run.

To find the top performing configurations for step s we look at the outcome of the learned model for each configuration, i.e. we subsequently take the rows of the pointwise performance matrix P_p into account and apply either the Friedman test [Friedman, 1937] for regression experiments or the Cochran's Q test [Cochran, 1950] to see whether we observe statistically significant differences between configurations (see Appendix A.4 for a summary of these tests).

More formally, the function TOPCONFIGURATIONS takes the pointwise performance matrix P_p as input and rearranges the rows according to the mean performances of the configurations yielding a matrix $\widetilde{P}_p$. Now for $k \in \{2, 3, \ldots, K\}$ we check, whether the first k configurations show a significantly different effect on the $N - n$ data points. This is done by executing either the Friedman test or the Cochran's Q test on the submatrix $\widetilde{P}_p[1 : k, 1 : (N - n)]$ with the pre-specified significance level α. If the test does not indicate a significant difference in the performance of the k configurations, we increment k by one and test again until we find a significant effect. Suppose we find a significant effect at index $\tilde{k}$. Since all previous tests indicated no significant effect for the $\tilde{k} - 1$ configurations we argue, that the addition of the $\tilde{k}^{\text{th}}$ configuration must have triggered the test procedure to

Algorithm 2 CVST Main Loop

1: **function** CVST($d_1, \ldots, d_N$, S, C, α, β_l, α_l, w_{stop})
2: $\Delta \leftarrow \text{N}/S$ ▷ Initialize subset increment
3: $n \leftarrow \Delta$ ▷ Initialize model size
4: test ←getTest(S, β_l, α_l) ▷ Get sequential test
5: $\forall \text{s} \in \{1, \ldots, S\}, c \in C : T_S[c, s] \leftarrow 0$
6: $\forall \text{s} \in \{1, \ldots, S\}, c \in C : P_S[c, s] \leftarrow \text{NA}$
7: $\forall c \in C : \text{isActive}[c] \leftarrow$**true**
8: **for** $s \leftarrow 1$ **to** S **do**
9: $\forall \text{i} \in \{1, \ldots, N - n\}, c \in C : P_p[c, i] \leftarrow \text{NA}$ ▷ Initialize pointw. perf.
10: **for** $c \in C$ **do**
11: **if** isActive[c] **then**
12: $g = g_n(c)$ ▷ Learn model on the first n data points
13: $\forall i \in \{1, \ldots, N - n\} : P_p[c, i] \leftarrow \ell(g(x_{n+i}), y_{n+i})$ ▷ Eval. on rest
14: $P_S[c, s] \leftarrow \frac{1}{N-n} \sum_{i=1}^{N-n} P_p[c, i]$ ▷ Store mean performance
15: $\text{index}_{\text{top}} \leftarrow$topConfigurations($P_p, \alpha$) ▷ Find the top configurations
16: $T_S[\text{index}_{\text{top}}, s] \leftarrow 1$ ▷ And set entry in trace matrix
17: **for** $c \in C$ **do**
18: **if** isActive[c] **and** isFlopConfiguration(test, $T_S[c, 1 : s]$) **then**
19: isActive[c] ←**false** ▷ De-activate flop configuration
20: **if** similarPerformance(T_S[isActive, $(s - w_{\text{stop}} + 1) : s$], α) **then**
21: **break**
22: $n \leftarrow n + \Delta$
23: **return** selectWinnner(P_S, isActive, w_{stop}, s)

indicate that in the set of these $\tilde{k}$ configurations is at least one configuration, which shows a significantly different behavior than all other configurations. Thus, we flag the configurations $1, \ldots, \tilde{k} - 1$ as top configurations and the remaining $\tilde{k}, \ldots, K$ configurations as flop configurations. Note that this incremental procedure is not a multiple testing situation, since we are not interested in a joint inference over all hypotheses but use each individual test to decide whether we can observe a significant effect due to the addition of a new configuration.

Note that for the actual calculation of the test errors we apply an incremental model building process, i.e., the data added in each step on Line 22 increases the training data pool for each step by a set of size Δ. This would allow online algorithms to adapt their model also incrementally leading to even further speed improvements. The results of this first step are collected for each configuration in the trace matrix T_S (see Figure 3.5, top right), which shows the gradual transformation for the last 10 steps of the procedure highlighting the results of the last test. So the robust transformation of the test error boils down the performance of all models learned on the first n data points to a new column in the trace matrix T_S recording the history of each configuration in a top or flop scheme.

3.2.2 Determining Significant Losers

Having transformed the test errors in a scale-independent top or flop scheme, we can now test whether a given parameter configuration is an overall loser. Sequential testing of binary random variables is addressed in the *sequential analysis* framework developed by Wald [1947]. Originally it has been applied in the context of production quality assessment (compare two production processes) or biological settings (stop bioassays as soon as the gathered data leads to a significant result).

The main idea is the following: One observes a sequence of i.i.d. Bernoulli variables $B_1, B_2, \ldots$, and wants to test whether these variables are distributed according to the hypotheses $H_0 : B_i \sim \pi_0$ or the alternative hypotheses $H_1 : B_i \sim \pi_1$ with $\pi_0 < \pi_1$ denoting the according success probabilities of the Bernoulli variables. Both significance levels for the acceptance of H_1 and H_0 can be controlled via the user-supplied meta-parameters α_l and β_l. The test computes the likelihood for the so far observed data and rejects one of the hypothesis when the respective likelihood ratio exceeds an interval controlled by the meta-parameters. It can be shown that the procedure has a very intuitive geometric representation, shown in Figure 3.5, lower left: The binary observations are recorded as cumulative sums at each time step. If this sum exceeds the upper red line L_1, we accept H_1; if the sum is below the lower red line L_0 we accept H_0; if the sum stays between the two red lines we have to draw another sample.

Wald's test requires that we fix both success probabilities π_0 and π_1 beforehand. Since our main goal is to use the sequential test to eliminate underperformers, we choose the parameters π_0 and π_1 of the test such that H_1 (a configuration wins) is postponed as long as possible. This will allow the CVST algorithm to keep configurations until the evidence of their performances definitely shows that they are overall loser configurations. At the same time, we want to maximize the area where configurations are eliminated (region $\triangle_{H_0}$ denoted by "LOSER" in Fig. 3.5), rejecting as many loser configurations on the way as possible:

$$\begin{aligned}(\pi_0, \pi_1) \quad = \quad & \underset{\pi_0', \pi_1'}{\operatorname{argmax}} \triangle_{H_0}(\pi_0', \pi_1', \beta_l, \alpha_l) \\ & \text{s.t. } S_a(\pi_0', \pi_1', \beta_l, \alpha_l) \in (S-1, S]\end{aligned} \tag{3.2}$$

with $S_a(\cdot, \cdot, \cdot, \cdot)$ is the earliest step of acceptance of H_1 marked by an X in Fig. 3.5 and the variable S defined as the total number of steps. Using results from Wald [1947] the global optimization in Equation (3.3) can be solved as follows:

$$\pi_0 = 0.5 \wedge \pi_1 = \min_{\pi_1'} \mathrm{ASN}(\pi_0, \pi_1' | \pi = 1.0) \geqslant S \tag{3.3}$$

where $\mathrm{ASN}(\cdot, \cdot)$ (Average Sample Number) is the expected number of steps until the given test will yield a decision, if the underlying success probability of the tested sequence is $\pi = 1.0$. Equipped with this test, we can check each remaining trace on Line 18 of Algorithm 2 in the function isFlopConfiguration whether it is a statistically significant flop configuration (i.e. exceeds the lower decision boundary L_0) or not.

Note that sequential analysis formally requires i.i.d. variables, which might not be true for configurations which transform to a winner configuration later on, thereby changing their behavior from a flop to a top configuration. Therefore we tuned our procedure to use the sequential analysis framework just for the decision whether a configuration is an overall loser or not. The test is adjusted for this switch of roles by keeping potential configurations as long as possible and just drop them if its trace statistical significantly corresponds to a binomial with $\pi < 0.5$. For details of the open sequential analysis consult Wald [1947] or see for instance Wetherill and Glazebrook [1986] for a general overview of sequential testing procedures. Appendix B.4 contains the necessary details needed to implement the proposed testing scheme for the CVST algorithm.

3.2.3 Early Stopping and Final Winner

Finally, we employ an early stopping rule (Line 20) which takes the last w_{stop} columns from the trace matrix and checks whether all remaining configurations performed equally well in the past. In Figure 3.5 this submatrix of the overall trace matrix T_S is shown for a value of $w_{\text{stop}} = 4$ for the remaining configurations after step 10. For the test, we again apply the Cochran's Q test (see Appendix A.4) in the similarPerformance procedure on the submatrix of T_S. Figure 3.6 illustrates a complete run of the CVST algorithm for roughly 600 configurations. Each configuration marked in red corresponds to a flop configuration and a black one to a top configuration. Finally, configurations marked in gray have been dropped via the sequential test during the CVST algorithm. The small zoom-ins in the lower part of the picture show the last w_{stop} remaining configurations during each step which are used in the evaluation of the early stopping criterion. We can see that the procedure keeps on going if there is a heterogeneous behavior of the remaining configurations (zoom-in is mixed red/black). When the the remaining configurations all performed equally well in the past (zoom-in is nearly black), the early stopping test does not see a significant effect anymore and the procedure is stopped.

Finally, in the procedure selectWinner, Line 23, the winning configuration is picked from the configurations which have survived all steps as follows: For each remaining configuration we determine the rank in a step according to the average performance during this step. Then we average the rank over the last w_{stop} steps and pick the configuration which has the lowest mean rank. This way, we make most use of the data accumulated during the course of the procedure. By restricting our view to the last w_{stop} observations we also take into account that the optimal parameter might change with increasing model size: since we focus on the most recent observations with the biggest models, we always pick the configuration which is most suitable for the data size at hand.

3.2.4 Meta Parameters for the CVST

The CVST algorithm has a number of meta parameters which the experimenter has to determine beforehand. In this section we give suggestions how to choose these parameters. The parameter α controls the significance level of the test for

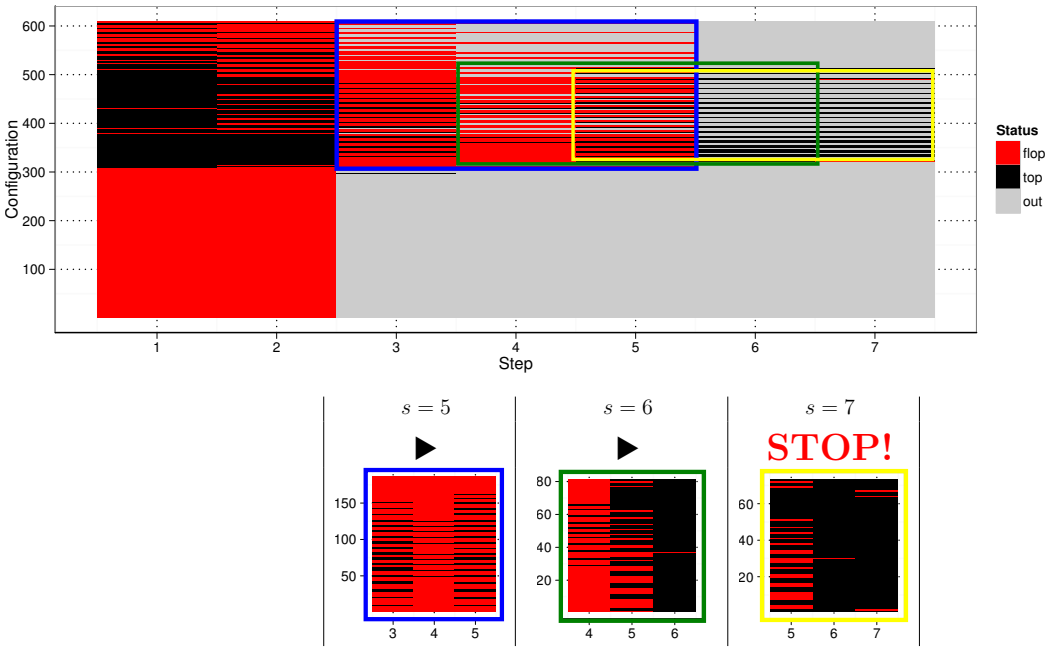


Fig. 3.6: The upper plot shows a run of the CVST algorithm for roughly 600 configurations. At each step a configuration is marked as top (black), flop (red) or dropped (gray). The zoom-ins show the situation for step 5 to 7 without the dropped entries. The early stopping rule takes affect in step 7, because the remaining configurations performed equally well during step 5 to 7.

similar behavior in each step of the procedure. We suggest to set this to the usual level of $\alpha = 0.05$. Furthermore β_l and α_l control the significance level of the H_0 (configuration is a loser) and H_1 (configuration is a winner) respectively. We suggest an asymmetric setup by setting $\beta_l = 0.1$, since we want to drop loser configurations relatively fast and $\alpha_l = 0.01$, since we want to be really sure when we accept a configuration as overall winner. Finally, we set w_{stop} to 3 for $S = 10$ and 6 for $S = 20$, as we have observed that this choice works well in practice.

3.3 Theoretical Properties of the CVST Algorithm

After having introduced the overall concept of the CVST algorithm, we now focus on the theoretical properties, which ensure the proper working of the procedure: Exploiting guarantees of the underlying sequential testing framework, we show how the experimenter can control the procedure to work in a stable regime and furthermore prove error bounds for the CVST algorithm. Additionally, we show how the CVST algorithm can be used to work best on a given time budget.

3.3.1 Error Bounds in a Stable Regime

As discussed in Section 3.1 the performance of a configuration might change if we feed the learning algorithm more data. Therefore, a reasonable algorithm exploiting the learning on subsets of the data must be capable of dealing with these difficulties and potential change points in the behavior of certain configurations. In this section we investigate some theoretical properties of the CVST algorithm which makes it particularly suitable for learning on increasing subsets of the data.

The first property of the open sequential test employed in the CVST algorithm comes in handy to control the overall convergence process and to assure that no configurations are dropped prematurely:

Lemma 2 (Safety Zone) *Given the CVST algorithm with significance level α_l, β_l for being a top or flop configuration respectively, and maximal number of steps S, and a global winning configuration, which looses for the first s_{cp} iterations, as long as*

$$0 \leqslant \frac{s_{cp}}{S} \leqslant \frac{s_{safe}}{S} \text{ with } s_{safe} = \frac{\log \frac{\beta_l}{1-\alpha_l}}{\log 2 - \sqrt[S]{\frac{1-\beta_l}{\alpha_l}}} \text{ and } S \geqslant \left\lceil \log \frac{1-\beta_l}{\alpha_l} / \log 2 \right\rceil,$$

the probability that the configuration is dropped by the CVST algorithm is zero.

Proof The details of the proof are deferred to Appendix B.4. ■

The consequence of Lemma 2 is that the experimenter can directly control via the significance levels α_l, β_l until which iteration no premature dropping should occur and therefore guide the whole process into a stable regime in which the configurations will see enough data to show their real performance.

Equipped with this property we can now take a thorough look at the worst case performance of the CVST algorithm: Suppose a global winning configuration has been constantly marked as a loser up to the safety zone, because the amount of data available up to this point was not sufficient to show the superiority of this configuration. Given that the global winning configuration now sees enough data to be marked as a winning configuration by the binarization process throughout the next steps with probability π we can give exact error bounds of the overall process by solving specific recurrences.

Figure 3.7 gives a visual impression of our worst case analysis for the example of a 20 step CVST execution: The winning configuration generated a straight line of zeros up to the safety zone of 7. Our approach to bound the error of the fast cross-validation now consists essentially in calculating the probability mass that ends up in the non-loser region. The following lemma shows how we can express the number of paths which lead to a specific point on the graph by a two-dimensional recurrence relation:

Lemma 3 (Recurrence Relation) *Denote by $\mathcal{P}(s_R, s_C)$ the number of paths, which lead to the point at the intersection of row s_R and column s_C and lie above*

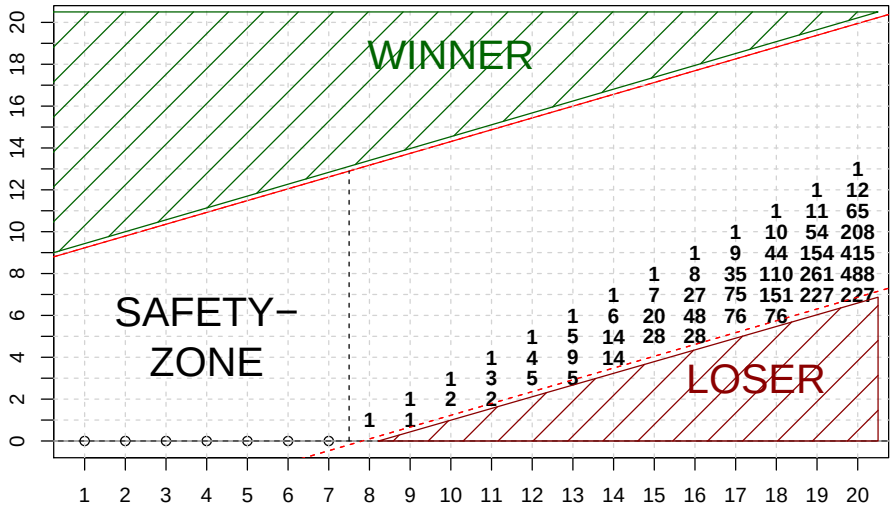


Fig. 3.7: Visualization of the worst-case scenario for the error probability of the CVST algorithm: a global winner configuration is labeled as a constant loser until the safety zone is reached. Then we can calculate the probability that this configuration endures the sequential test by a recurrence scheme, which counts the number of remaining paths ending up in the non-loser region.

the lower decision boundary L_0 of the sequential test. Given the worst case scenario described above the number of paths can be calculated as follows:

$$\mathcal{P}(s_R, s_C) = \begin{cases} 1 & \text{if } s_R = 0 \wedge c \leqslant s_{safe} = \frac{\log \frac{\beta_l}{1-\alpha_l}}{\log 2 - \sqrt[S]{\frac{1-\beta_l}{\alpha_l}}} \\ 1 & \text{if } s_R = s_C - s_{safe} \\ \mathcal{P}(s_R, s_C - 1) + \mathcal{P}(s_R - 1, s_C - 1) & \text{if } L_0(c) < s_R < s_C - s_{safe} \\ 0 & \text{otherwise.} \end{cases}$$

Proof We split the proof into the four cases:

1. The first case is by definition: the configuration has a straight line of zeros up to the safety zone s_{safe}.

2. The second case describes the diagonal path starting from the point $(1, s_{\text{safe}} + 1)$: by construction of the paths (1 means diagonal up; 0 means one step to the right) the diagonal path can just be reached by a single combination, namely a straight line of ones.

3. The third case is the actual recurrence: if the given point is above the lower decision bound L_0, then the number of paths leading to this point is equal to the number of paths that lie directly to the left of this point plus the paths which lie directly diagonal downwards from this point. From the first paths

this point can be reached by a direct step to the right and from the latter the current point can be reached by a diagonal step upwards. Since there are no other options than that by construction, this equality holds.

4. The last case describes all other paths, which either lie below the lower decision bound and therefore end up in the loser region or are above the diagonal and thus can never be reached.

■

This recurrence is visualized in Figure 3.7. Each number on the grid gives the number of valid, non-loser paths, which can reach the specific point. With this recurrence we are now able to prove a global, worst-case error probability of the fast cross-validation.

Theorem 4 (Error Bound of CVST) *Suppose a global winning configuration has reached the safety zone with a constant loser trace and then switches to a winner configuration with a success probability of π. Then the error that the CVST algorithm erroneously drops this configuration can be determined as follows:*

$$P(\textit{reject } \pi) \leqslant 1 - \sum_{i=\lfloor L_0(S) \rfloor + 1}^{r} \mathcal{P}(i, S)\pi^i(1-\pi)^{r-i} \qquad \textit{with } r = S - \left\lfloor \frac{\log \frac{\beta_l}{1-\alpha_l}}{\log 2 - \sqrt[S]{\frac{1-\beta_l}{\alpha_l}}} \right\rfloor$$

Proof The basic idea is to use the number of paths leading to the non-loser region to calculate the probability that the configuration actually survives. This corresponds to the last column of the example in Figure 3.7. Since we model the outcome of the binarization process as a binomial variable with the success probability of π, the first diagonal path has a probability of π^r. The next paths each have a probability of $\pi^{(r-1)}(1-\pi)^1$ and so on until the last viable paths are reached in the point $(\lfloor L_0(S) \rfloor + 1, S)$. So the complete probability of the survival of the configuration is summed up with the corresponding number of paths from Lemma 3. Since we are interested in the complementary event, we subtract the resulting sum from one, which concludes the proof. ■

Note that the early stopping rule does not interfere with this bound: The worst case is indeed that the process goes on for the maximal number of steps S, since then the probability mass will be maximally spread due to the linear lower decision boundary and the corresponding exponents are maximal. So if the early stopping rule terminates the process before reaching the maximum number of steps, the resulting error probability will be lower than our given bound.

The error bound for different success probabilities and the proposed sequential test with $\alpha_l = 0.01$ and $\beta_l = 0.1$ are depicted in Figure 3.8. First of all we can observe a relative fast convergence of the overall error with increasing maximal number of steps S. The impact on the error is marginal for the shown success probabilities, i.e. for instance for $\pi = 0.95$ the error nearly converges to the optimum of 0.05. Note, that the oscillations especially for small step sizes originate

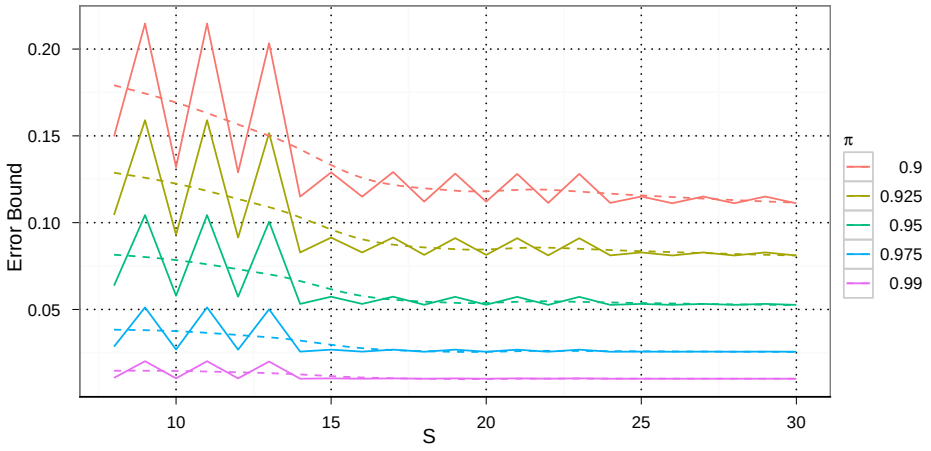


Fig. 3.8: Error bound of the fast cross-validation as proven in Theorem 4 for different success probabilities π and maximal step sizes S. To mark the global trend we fitted a LOESS curve given as dotted line to the data.

from the rectangular grid imposed by the interplay of the $\mathcal{P}(\cdot)$-operator and the lower decision boundary L_0 leading to some fluctuations. Overall, the chosen test scheme allows us not only to control the safety zone but also has only a small impact on the error probability, which once again shows the practicality of the open sequential ratio test for the fast cross-validation procedure. By using this statistical test we can balance the need for a conservative retention of configurations as long as possible with the statistically controlled dropping of significant loser configurations with nearly no impact on the overall error probability. Our analysis assumes that the experimenter has chosen the right safety zone for the learning problem at hand. For small data sizes it could happen that this safety zone was chosen too small, therefore the change point of the global winning configuration might lie outside the safety zone. While this will not occur often for today's sizes of data sets we have analyzed the behavior of CVST under this circumstances in Appendix B.5 to give a complete view of the properties of the algorithm.

3.3.2 Fast-Cross Validation on a Time Budget

While the CVST algorithm can be used out of the box to speed up regular cross-validation, the aforementioned properties of the procedure come in handy when we face a situation in which an optimal parameter configuration has to be found given a fixed computational budget. If the time is not sufficient to perform a full cross-validation or the amount of data that has to be processed is too big to explore a sufficiently spaced parameter grid with ordinary cross-validation in a reasonable time, the CVST algorithm allows for getting the most model selection information out of the data given the specified constraints.

This is achieved by calculating a maximal steps parameter S which leads to a near coverage of the available time budget T as depicted in Figure 3.9. The idea

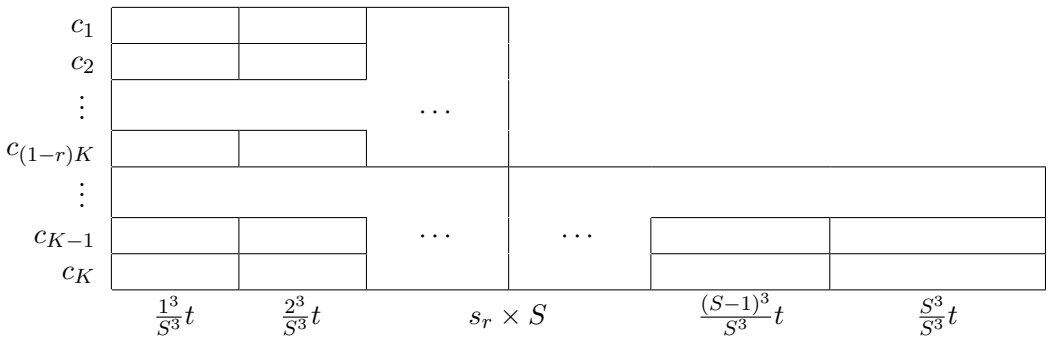


Fig. 3.9: Approximation of the time consumption for a cubic learner. In each step we calculate a model on a subset of the data, so the model calculation time t on the full data set is adjusted accordingly. After $s_r \times S$ steps of the process, we assume a drop to $r \times K$ remaining configurations.

is to specify an expected drop ratio r of configurations and a safety zone bound s_{safe}. Then we can give a rough estimate of the total time needed for a CVST with a total number of steps S, equating this with the available time budget T and solving for S. More formally, given K parameter configurations and a pre-specified safety zone bound $s_{\text{safe}} = s_r \times S$ with $0 < s_r < 1$ to ensure that no configuration is dropped prematurely, the computational demands of the CVST algorithm are approximated by the sum of the time needed before step s_{safe} involving the model calculation of all K configurations and after step s_{safe} for $r \times K$ configurations with $0 < r < 1$. As we will see in the experimental evaluation section, this assumption of a given drop rate of $(1-r)$ leading to the form of time consumption as depicted in Figure 3.9 is quite common. The observed drop rate corresponds to the overall difficulty of the problem at hand.

Given the computation time t needed to perform the model calculation on the full data set, we prove in Appendix B.6 that the optimal maximum step parameter for a cubic learner can be calculated as follows:

$$S = \left\lfloor \frac{2T - Kt(1-r)s_r^3 - rKt}{((1-r)s_r^4 + r)Kt} + \sqrt{\left[\frac{Kt(1-r)s_r^3 + rKt - 2T}{((1-r)s_r^4 + r)Kt}\right]^2 - \frac{(1-r)s_r^2 + r}{(1-r)s_r^4 + r}} \right\rfloor$$

After calculating the maximal number of steps S given the time budget T, we can use the results of Lemma 2 to determine the maximal β_l given a fixed α_l, which yields the requested safety zone bound s_{safe}.

3.4 Experiments

Before we evaluate the CVST algorithm on real data, we investigate its performance on controlled data sets. Both for regression and classification tasks we introduce special tailored data sets to highlight the overall behavior and to stress test the fast cross-validation procedure. To evaluate how the choice of learning

method influences the performance of the CVST algorithm, we compare kernel logistic regression (KLR) against a ν-support vector machine (SVM) for classification problems and kernel ridge regression (KRR) versus ν-SVR for regression problems each using a Gaussian kernel [see Roth, 2001, Schölkopf et al., 2000]. In all experiments we use a 10 step CVST with parameter settings as described in Section 3.2.4 (i. e. $\alpha = 0.05, \alpha_l = 0.01, \beta_l = 0.1, w_{\text{stop}} = 3$) to give us an upper bound of the expected speed gain. Note that we could get even higher speed gains by either lowering the number of steps or increasing β_l. From a practical point of view we believe that the settings studied are highly realistic.

3.4.1 Artificial Data Sets

To assess the quality of the CVST algorithm we first examine its behavior in a controlled setting. We have seen in our motivation section that a specific learning problem might have several layers of structure which can only be revealed by the learner if enough data is available. For instance in Figure 3.4a we can see that the first optimal plateau occurs at $\sigma = 0.1$, while the real optimal parameter centers around $\sigma = 0.01$. Thus, the real optimal choice just becomes apparent if we have seen more than 200 data points.

In this section we construct a learning problem both for regression and classification tasks which could pose severe problems for the CVST algorithm: If it stops too early, it will return a suboptimal parameter set. We evaluate how different intrinsic dimensionalities of the data and various noise levels affect the performance of the procedure. For classification tasks we use the *noisy sine* data set, which consists of a sine uniformly sampled from a range controlled by the intrinsic dimensionality d:

$$y = \sin(x) + \epsilon \quad \text{with } \epsilon \sim \mathcal{N}(0, n^2), x \in [0, 2\pi d], \\ n \in \{0.25, 0.5\}, d \in \{5, 50, 100\}$$

The labels of the sampled points are just the sign of y. For regression tasks we devise the *noisy sinc* data set, which consists of a sinc function overlayed with a high-frequency sine:

$$y = \operatorname{sinc}(4x) + \frac{\sin(15dx)}{5} + \epsilon \quad \text{with } \epsilon \sim \mathcal{N}(0, n^2), x \in [-\pi, \pi], \\ n \in \{0.1, 0.2\}, d \in \{2, 3, 4\}$$

For each of these data sets we generate 1,000 data points and run a 10 step CVST and compare its results with a normal 10-fold cross-validation on the full data set. We record both the test error on additional 10,000 data points and the time consumed for the parameter search. The explored parameter grid contains 610 equally spaced parameter configurations for each method ($\log_{10}(\sigma) \in \{-3, -2.9, \ldots, 3\}$ and $\nu \in \{0.05, 0.1, \ldots, 0.5\}$ for SVM/SVR and $\log_{10}(\lambda) \in \{-7, -6, \ldots 2\}$ for KLR/KRR, respectively). This process is repeated 50 times to gather sufficient data for an interpretation of the overall process.

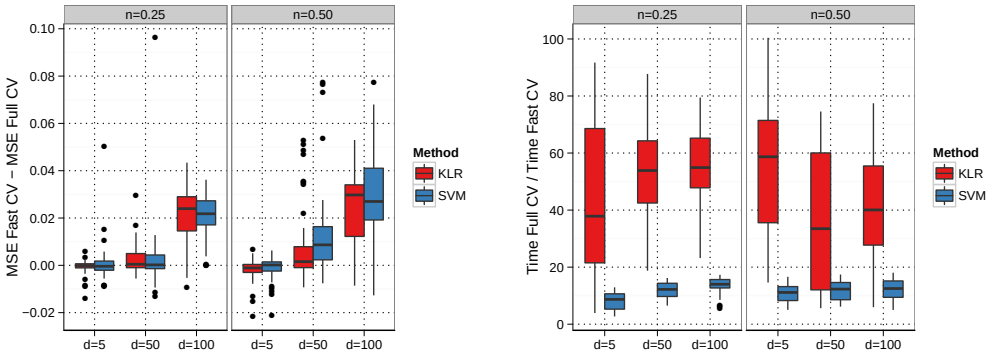


Fig. 3.10: Difference in mean square error (left) and relative speed gain (right) for the *noisy sine* data set.

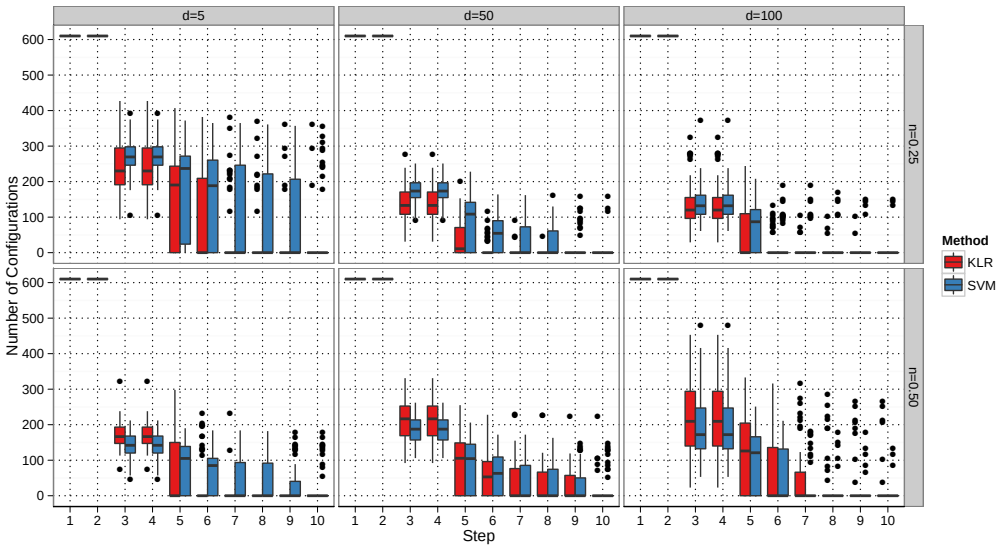


Fig. 3.11: Remaining configurations after each step for the *noisy sine* data set.

The results for the *noisy sine* data set can be seen in Figure 3.10. The upper boxplots show the distribution of the difference in mean square error of the best parameter determined by CVST and normal cross-validation. In the low noise setting ($n = 0.25$) the CVST algorithm finds the same optimal parameter as the normal cross-validation up to the intrinsic dimensionality of $d = 50$. For $d = 100$ the CVST algorithm gets stuck in a suboptimal parameter configuration yielding an increased classification error compared to the normal cross-validation. This tendency is slightly increased in the high noise setting ($n = 0.5$) yielding a broader distribution. The classification method used seems to have no direct influence on the difference, both SVM and KLR show nearly similar behavior. This picture

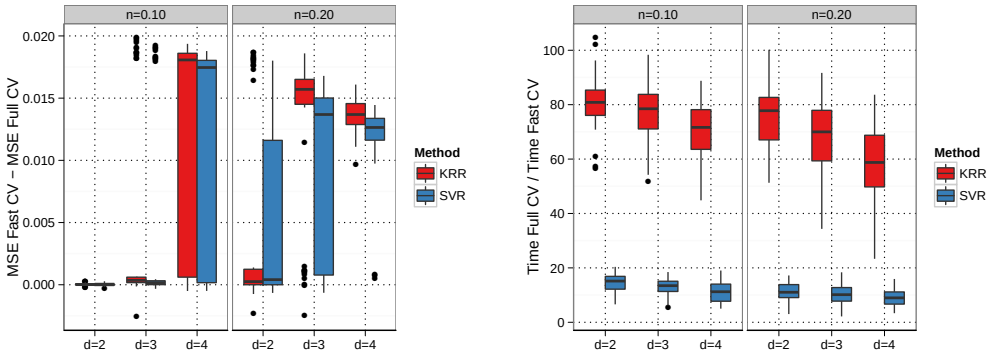


Fig. 3.12: Difference in mean square error (left plots) and relative speed gain (right plots) for the *noisy sinc* data set.

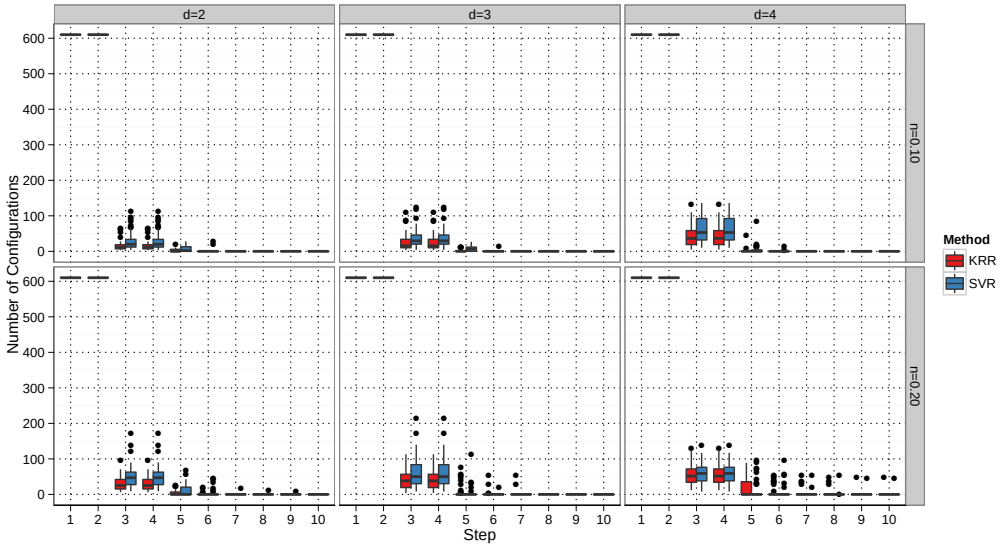


Fig. 3.13: Remaining configurations after each step for the *noisy sinc* data set.

changes when we look at the speed gains: While the SVM nearly always ranges between 15 and 19, the KLR shows a speed-up between 20 and 60 times. The variance of the speed gain is generally higher compared to the SVM which seems to be a direct consequence of the inner workings of KLR. Basically the main loop performs at each step a matrix inversion of the whole kernel matrix until the calculated coefficients converge. Obviously this convergence criterion leads to a relative wide-spread distribution of the speed gain when compared to the SVM performance.

Figure 3.11 shows the distribution of the number of remaining configurations after each step of the CVST algorithm. In the low noise setting (upper row) we

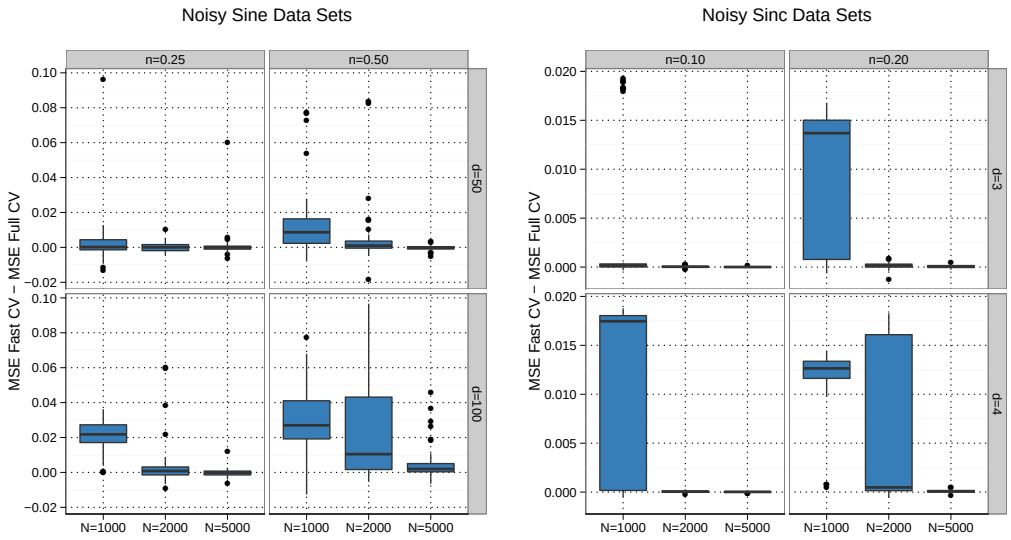


Fig. 3.14: Difference in mean square error for SVM/SVR with increasing data set size for *noisy sine* (left) and the *noisy sinc* (right) data sets. By adding more data, the CVST algorithm converges to the correct parameter configuration.

can observe a tendency of bigger dropping rates up to $d = 100$. For the high noise setting (lower row) we observe a steady increase of kept configurations combined with a higher spread of the distribution. Overall we see a very effective dropping rate of configurations for all settings. The SVM and the KLR show nearly similar behavior so that the higher speed gain of the KLR we have seen before is a direct consequence of the algorithm itself and is not influenced by the CVST algorithm.

The performance on the *noisy sinc* data set is shown in Figure 3.12. The first striking observation is the transition of the CVST algorithm which can be observed for the intrinsic dimensionality of $d = 4$. At this point the overall excellent performance of the CVST algorithm is on the verge of choosing a suboptimal parameter configuration. This behavior is more evident in the high noise setting. The SVR nearly always shows a smaller difference than the KRR and is capable of delaying the decline in the high noise setting at least partly. The speed gain observed is nearly constant over the different dimensionalities and noise levels and ranges between 15 and 20 for the SVR and 60 to 80 for KRR.

This is a direct consequence of the behavior which can be observed in the number of remaining configurations shown in Figure 3.13. Compared to the classification experiments the drop is much more drastic. The intrinsic dimensionality and the noise level show a small influence (higher dimensionality or noise level yields more remaining configurations) but the overall variance of the distribution is much smaller than in the classification experiments.

In Figure 3.14 we examine the influence of more data on the performance of the CVST algorithm. Both for the *noisy sine* and *noisy sinc* data set we are able to estimate the correct parameter configuration for all noise and dimensionality

settings if we feed the CVST with enough data.[2] Clearly, the CVST is capable of extracting the right parameter configuration if we increase the amount of data to 2000 or 5000 data points, rendering our method even more suitable for big data scenarios: If data is abundant, CVST will be able to estimate the correct parameter in a much smaller time frame.

3.4.2 Benchmark Data Sets

After demonstrating the overall performance of the CVST algorithm on controlled data sets we will investigate its performance on real life and well-known benchmark data sets. For classification we pick a representative choice of data sets from the IDA benchmark repository [see Rätsch et al. 2001[3]] and add the static feature data set from the Cujo system [Rieck et al., 2010].[4] Furthermore we add the first two classes with the most entries of the *covertype* data set [see Blackard and Dean, 1999]. Then we follow the procedure of the paper in sampling 2,000 data points of each class for the model learning and estimate the test error on the remaining data points. For regression we pick the data used in Donoho and Johnstone [1994] and add the *bank32mn*, *pumadyn32mn* and *kin32mn* of the Delve repository.[5]

We process each data set as follows: First we normalize each variable of the data to zero mean and variance of one, and in case of regression we also normalize the dependent variable. Then we split the data set in half and use one part for training and the other for the estimation of the test error. This process is repeated 50 times to get sufficient statistics for the performance of the methods. As in the artificial data setting we compare the difference in test error and the speed gain of the fast and normal cross-validation on the same parameter grid of 610 values.[6]

Figure 3.15 shows the result for the classification data sets (left side) and the regression data sets (right side). The upper panels depict the difference in mean square error (MSE). For the classification tasks this difference never exceeds two percent points showing that although the fast cross-validation procedure in some cases seems to pick a suboptimal parameter set, the impact of this false decision is small. The same holds true for the regression tasks: since the dependent variables for all problems have been normalized to zero mean and variance of one, the differences in MSE values are comparable. We observe that as for the classification tasks we see just a very small difference in MSE. Although for some problems the CVST algorithm picks a suboptimal parameter set, even then the differences in error are always relatively small. The learners have hardly any impact on the behavior; just for the *covertype* and the *blocks* data set we see a significant difference of the corre-

[2]Note, that we have to limit this experiment to the SVM/SVR method, since the full cross-validation of the KLR/KRR would have taken to much time to compute.

[3]Available at http://www.mldata.org.

[4]To allow for efficient calculation we precomputed the kernel matrix and thus randomly picked just 20.000 instances of the benign scripts and all of the malicious ones.

[5]Available at http://www.cs.toronto.edu/~delve.

[6]For the *blocks*, *bumps*, and *doppler* data set of Donoho and Johnstone [1994] we had to adjust the range of σ to $\log_{10}(\sigma) \in \{-6, -5.9, \ldots, 0\}$ to adjust to the small structure found in these data sets. For the *cujo* data set of [Rieck et al., 2010] we used a linear kernel and expanded the grid to contain 2, 3, and 4 token n-grams.

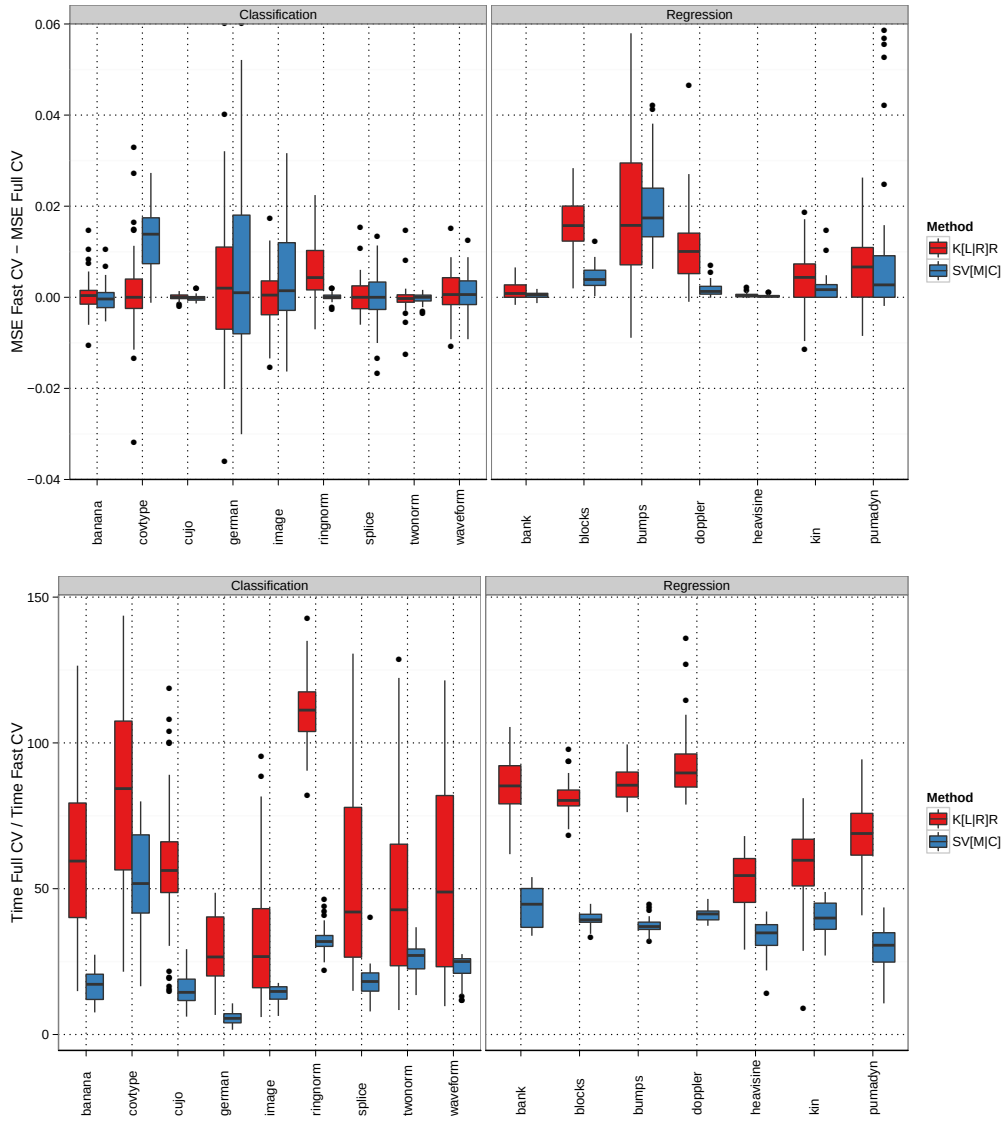


Fig. 3.15: Difference in mean square error (upper plots) and relative speed gain (lower plots) for the *benchmark* data sets.

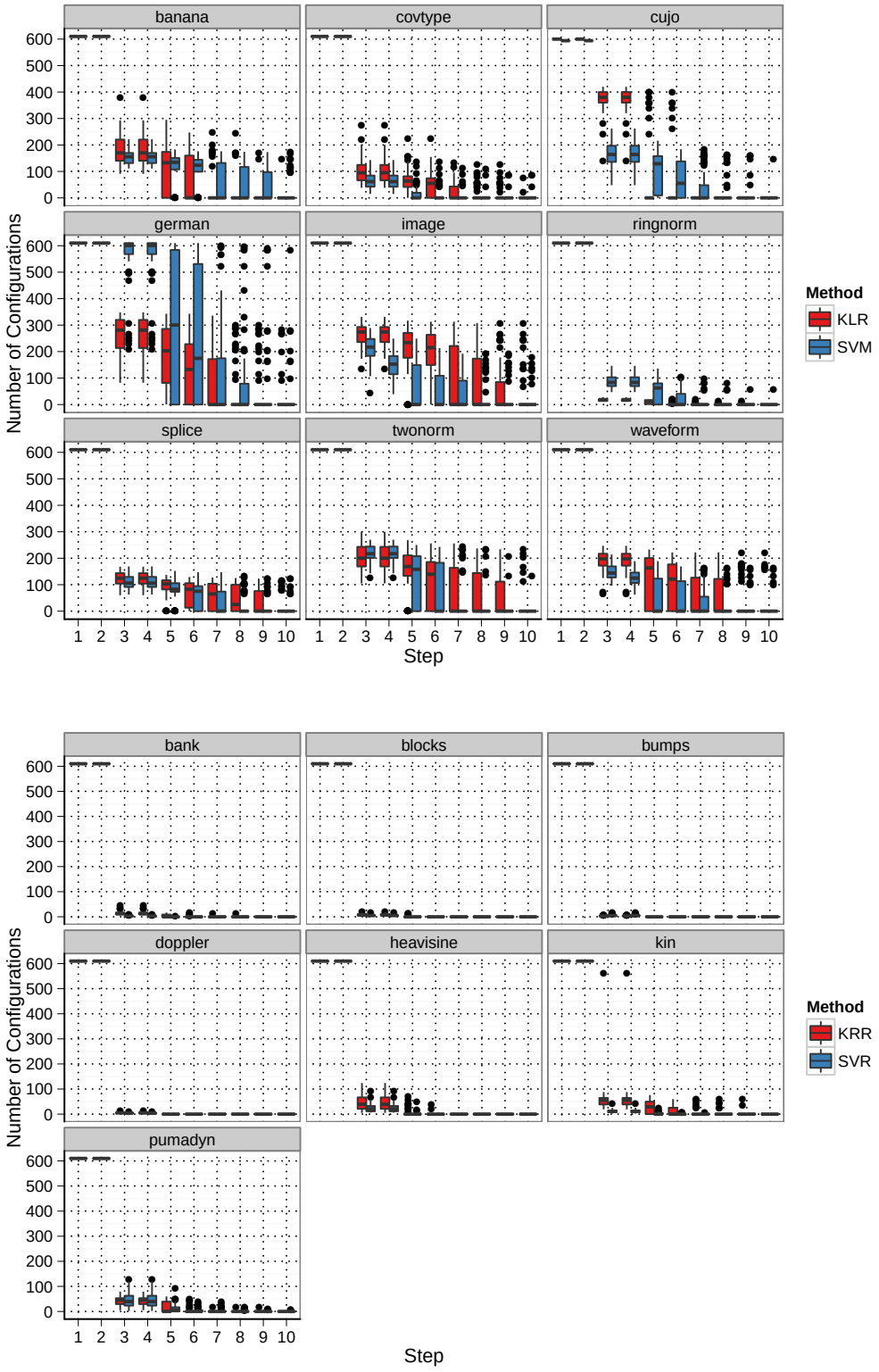


Fig. 3.16: Remaining configurations after each step for the *benchmark* data sets.

sponding methods. In terms of speed gain we see a much more diverse and varying picture. Overall, the speed improvements for KLR and KRR are higher than for SVM and SVR and reach up to 120 times compared to normal cross-validation.

Regression tasks in general seem to be solved faster than classification tasks, which can clearly be explained when we look at the traces in Figure 3.16: For classification tasks the number of kept configurations is generally much higher than for the regression tasks. Moreover we can observe different classes of difficulty for the learning tasks: The *german* data set seems to be much more difficult than the *ringnorm* data [see Braun et al., 2008] which is also reflected in the difference and speed improvement seen in the previous figure. The CVST method shows an overall good behavior on the *cujo* data: While the difference in MSE is practically zero, the speed improvements range between roughly 60 times faster for KLR and 15 times faster for the SVM showing the applicability of our method for this security problem.

In summary, the evaluation of the benchmark data sets shows that the CVST algorithm gives a huge speed improvement compared to the normal cross-validation. While we see some non-optimal choices of configurations, the total impact on the error is never exceptionally high. We have to keep in mind that we have chosen the parameters of our CVST algorithm to give an impression of the maximal attainable speed-up: more conservative setting would trade computational time for lowering the impact on the test error.

3.5 Discussion and Related Work

Sequential testing has been used extensively in the machine learning context. Section 3.5.1 summarizes the work related to the CVST algorithm. Section 3.5.2 discusses a variant of the sequential test and evaluates its performance for the CVST algorithm. It is shown that this so-called *closed* sequential test lacks essential properties of the open variant of Wald used in the CVST algorithm, which further underlines the optimality of the open test of Wald for the learning on increasing subsets of data.

3.5.1 Sequential Testing in Machine Learning

Using statistical tests and the sequential analysis framework in order to speed up learning has been the topic of several lines of research. However, the existing body of work mostly focuses on reducing the number of test evaluations, while we focus on the overall process of eliminating candidates themselves. To the best of our knowledge, this is a new concept and can apparently be combined with the already available racing techniques to further reduce the total calculation time.

Maron and Moore [1994, 1997] introduce the so-called *Hoeffding Races* which are based on the non-parametric Hoeffding bound for the mean of the test error. At each step of the algorithm a new test point is evaluated by all remaining models and the confidence intervals of the test errors are updated accordingly. Models whose confidence interval of the test error lies outside of at least one interval of a better performing model are dropped. Chien et al. [1995, 1999] devise a similar range of

algorithms using concepts of PAC learning and game theory: different hypotheses are ordered by their expected utility according to the test data the algorithm has seen so far. As for Hoeffding Races, the emphasis in this approach lies on reducing the number of evaluations.

This concept of racing is further extended by Domingos and Hulten [2001]: By introducing an upper bound for the learner's loss as a function of the examples, the procedure allows for an early stopping of the learning process, if the loss is nearly as optimal as for infinite data. Birattari et al. [2002] apply racing in the domain of evolutionary algorithms and extend the framework by using the Friedman test to filter out non-promising configurations. While Bradley and Schapire [2008] use similar concepts in the context of boosting (FilterBoost), Mnih et al. [2008] introduce the empirical Bernstein Bounds to extend both the FilterBoost framework and the racing algorithms. In both cases the bounds are used to estimate the error within a specific ϵ region with a given probability. Pelossof and Jones [2009] use the concept of sequential testing to speed up the boosting process by controlling the number of features which are evaluated for each sample. In a similar manner this approach is used in Pelossof and Ying [2010] to increase the speed of the evaluation of the perceptron and in Pelossof and Ying [2011] to speed up the Pegasos algorithm. Stanski [2012] and Stanski and Hellwich [2012] use a partial leave-one-out evaluation of model performance to get an estimate of the overall model performance, which then is used to pick the best model with highest probability. These racing concepts are applied in a wide variety of domains like reinforcement learning [Heidrich-Meisner and Igel, 2009] and timetabling [Birattari, 2009] showing the relevance and pratical impact of the topic.

Recently, Bayesian optimization has been applied to the problem of hyperparameter optimization of machine learning algorithms. Bergstra et al. [2011] use the sequential model-based global optimization framework (SMBO) and implement the loss function of an algorithm via hierarchical Gaussian processes. Given the previously observed history of performances, a candidate configuration is selected which minimizes this historical surrogate loss function. Applied to the problem of training deep belief networks this approach shows superior performance over random search strategies. Snoek et al. [2012] extend this approach by including timing information for each potential model, i.e. the cost of learning a model and optimizing the expected improvement per seconds leads to a global optimization in terms of wall-clock time. Thornton et al. [2012] apply the SMBO framework in the context of the WEKA machine learning toolbox: the so-called Auto-WEKA procedure does not only find the optimal parameter for a specific learning problem but also searches for the most suitable learning algorithm. Like the racing concepts, these Bayesian optimization approaches are orthogonal to the CVST approach and could be combined to speed up each step of the CVST loop.

On first sight, the multi-armed bandit problem [Berry and Fristedt, 1985, Cesa-Bianchi and Lugosi, 2006] also seems to be related to the problem here in another way: In the multi-armed bandit problem, a number of distributions are given and the task is to identify the distribution with the largest mean from a chosen sequence of samples from the individual distributions. In each round, the agent chooses one distribution to sample from and typically has to find some balance

between exploring the different distributions, rejecting distributions which do not seem promising and focusing on a few candidates to get more accurate samples.

This looks similar to our setting where we also wish to identify promising candidates and reject underperforming configurations early on in the process, but the main difference is that the multi-armed bandit setting assumes that the distributions are fixed whereas we specifically have to deal with distributions which change as the sample size increases. This leads to the introduction of a safety zone, among other things. Therefore, the multi-armed bandit setting is not applicable across different sample sizes. On the other hand, the multi-armed bandit approach is a possible extension to speed up the computation within a fixed training size, either in testing similar to the Hoeffding races already mentioned above, or to eliminate the computation of individual folds in the cross-validation procedure for underperforming configurations.

3.5.2 Open versus Closed Sequential Testing

As already introduced in Section 3.2.2 the sequential testing was pioneered by Wald [1947]; the test monitors a likelihood ratio of a sequence of i.i.d. Bernoulli variables $B_1, B_2, \ldots$:

$$\ell = \prod_{k=1}^{n} f(b_k, \pi_1) / \prod_{k=1}^{n} f(b_k, \pi_0) \quad \text{given } H_h : B_i \sim \pi_h, h \in \{0, 1\}.$$

Hypothesis H_1 is accepted if $\ell \geqslant A$ and contrary H_0 is accepted if $\ell \leqslant B$. If neither of these conditions apply, the procedure cannot accept either of the two hypotheses and needs more data. A and B are chosen such that the error probability of the two decisions does not exceed α_l and β_l respectively. In Wald and Wolfowitz [1948] it is proven that the open sequential probability ratio test of Wald is optimal in the sense, that compared to all tests with the same power it requires on average fewest observations for a decision. The testing scheme of Wald is called *open* since the procedure could potentially go on forever, as long as ℓ does not leave the (A, B)-tunnel.

The open design of Wald's procedure led to a development of different sequential tests, where the number of observations is fixed beforehand [see Armitage, 1960, Spicer, 1962, Alling, 1966, McPherson and Armitage, 1971]. For instance in clinical studies it might be impossible or ethically prohibitive to use a test which potentially could go on forever. Unfortunately, none of these so-called closed tests exhibit an optimality criterion, therefore we choose one which at least in simulation studies showed the best behavior in terms of average sample number statistics: The method of Spicer [1962] is based on a gambler's ruin scenario in which both players have a fixed fortune and decide to play for n games. If $f(n, \pi, F_a, F_b)$ is the probability that a player with fortune F_a and stake b will ruin his opponent with fortune F_b

in exactly n games, then the following recurrence holds:

$$f(n, \pi, F_a, F_b) = \begin{cases} 0 & \text{if } F_a < 0 \vee (n = 0 \wedge F_b > 0), \\ 1 & \text{if } n = 0 \wedge F_a > 0 \wedge F_b \leqslant 0, \\ \pi f(n-1, \pi, F_a + 1, F_b - b) & \\ \qquad +(1-\pi) f(n-1, \pi, F_a - b, F_b + b) & \text{otherwise.} \end{cases}$$

In each step, the player can either win a game with probability π and win 1 from his opponent or lose the stake b to the other player. Now, given $n = x + y$ games of which player A has won y and player B has won x, the game will stop if either of the following conditions hold:

$$y - bx = -F_a \Leftrightarrow y = \frac{b}{1+b} n - \frac{F_a}{1+b} \quad \text{or} \quad y - bx = F_b \Leftrightarrow y = \frac{b}{1+b} n - \frac{F_b}{1+b}.$$

This formulation casts the gambler's ruin problem into a Wald-like scheme, where we just observe the cumulative wins of player A and check whether we reached the lower or upper line. If we now choose F_a and F_b such that $f(n, 0.5, F_a, F_b) \leqslant \alpha_l$, we construct a test which allows us to check whether a given configuration performs worse than $\pi = 0.5$ (i.e. crosses the lower line) and can therefore be flagged as an overall loser with controlled error probability of α_l (see Alling [1966]). For more details on the closed design of Spicer please consult Spicer [1962].

Since simulation studies show that the closed variants of the sequential testing exhibit low average sample number statistics, we first have a look at the runtime performance of the CVST algorithm equipped with either the open or the closed sequential test. The most influential parameter in terms of runtime is the S parameter. In principle, a larger number of steps leads to more robust estimates, but also to an increase of computation time. We study the effect of different choices of this parameter in a simulation. For the sake of simplicity we assume that the binary top or flop scheme consists of independent Bernoulli variables with $\pi_{\text{winner}} \in [0.9, 1.0]$ and $\pi_{\text{loser}} \in [0.0, 0.1]$. We test both the open and the closed sequential test and compare the relative speed-up of the CVST algorithm compared to a full 10-fold cross-validation in case the learner is cubic.

Figure 3.17 shows the resulting simulated runtimes for different settings. The overall speed-up is much higher for the closed sequential test indicating a more aggressive behavior compared to the more conservative open alternative. Both tests show their highest increase in the range of 10 to 20 steps with a rapid decline towards the higher step numbers. So in terms of speed the closed sequential test definitely beats the more conservative open test.

Figure 3.18 reveals that the speed gain comes at a price: Apart from having no control over the safety zone, the number of falsely dropped configurations is much higher than for the open sequential test. While having a definitive advantage over the open test in terms of speed, the false negative rate of the closed test renders it useless for the CVST algorithm.

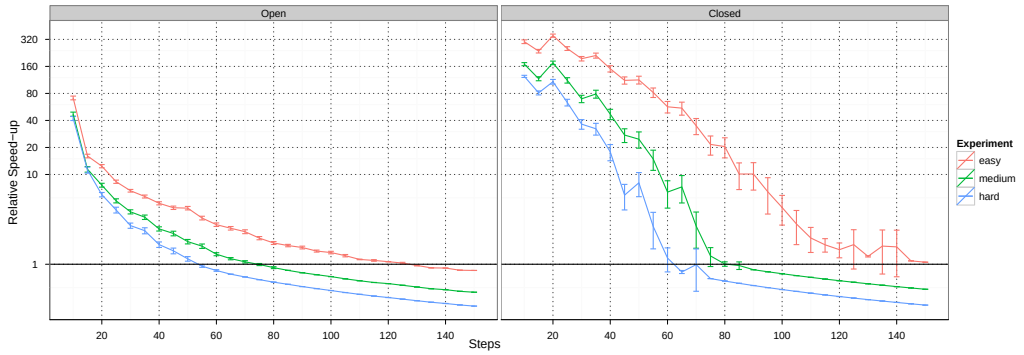


Fig. 3.17: Relative speed gain of fast cross-validation compared to full cross-validation. We assume that training time is cubic in the number of samples. Shown are simulated runtimes for 10-fold cross-validation on different problem classes by different loser/winner ratios (easy: 3:1; medium: 1:1, hard: 1:3) over 200 resamples.

3.6 Outlook and Conclusion

We presented a method to speed up the cross-validation procedure by starting at subsets of the full training set size, identifying clearly underperforming parameter configurations early on and focus on the most promising candidates for the larger subset sizes. We have discussed that taking subsets of the data set has theoretical advantages when compared to other heuristics like local search on the parameter set because the effects on the test errors are systematic and can be understood statistically. On the one hand, we showed that the optimal configurations converge to the true ones as sample sizes tend to infinity, but we also discussed in a concrete setting how the different behaviors of estimation error and approximation error lead to much faster convergence practically. These insights led to the introduction of a safety zone through sequential testing, which ensures that underperforming configurations are not removed prematurely when the minima are not converged yet. In experiments we showed that our procedure leads to a speed-up of up to 120 times compared to the full cross-validation without a significant increase in prediction error.

It will be interesting to combine this method with other procedures like the Hoeffding races or algorithms for multi-armed bandit problems. Furthermore, getting accurate convergence bounds even for finite sample size settings is another topic for future research. At the moment, the CVST algorithm has as a precondition that the optimal parameter of the learning method is independent of the training set size. While most of the standard machine learning algorithms adhere to this precondition there are other methods like k-nearest neighbor which exhibit a connection between the optimal parameter and the training set size. It would be a valuable extension of the CVST to incorporate this kind of knowledge into the algorithm. This would lower the preconditions on the learner and make the fast cross-validation procedure available to more learning methods.

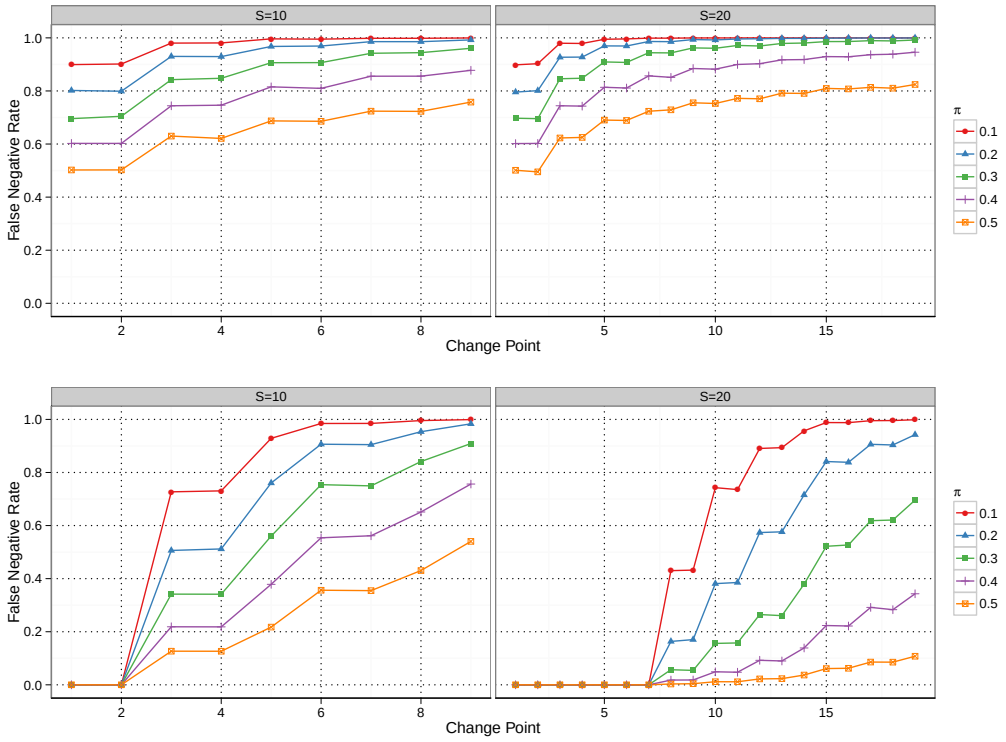


Fig. 3.18: False negatives generated with the closed (top) and open (bottom) sequential test for non-stationary configurations, i.e., at the given change point the Bernoulli variable changes its π_{before} from the indicated value to 1.0.

From a structural point of view we can see that the problem of selecting the optimal learner for a given problem can be deconstructed and decoupled into several layers: First, we define the problem domain of model complexity to be solved via the expected risk of a learner. Exploiting the convergence of the risk on subsets we are able to represent the performance of a learner in a binary fashion whether it belongs to the top performers of a round or not. By observing this property for increasing subset sizes we can give statistically bounded guarantees on the overall performance of a model via sequential testing and even use robust testing for early stopping of the whole procedure in case no more changes can be expected. This layering of probabilistic preprocessing and modeling allows for a step-by-step construction process which combined in one tool-chain solves the problem of model selection. This toolbox is publicly available as an open source CRAN package (see R Core Team [2012]) named `CVST`. Apart from the CVST algorithm itself it contains all the learning algorithms used in this chapter readily packaged for the use in the fast cross-validation framework. Thus, CVST can now easily be used as an additional encapsulated layer in other learning tasks to facilitate and speed up the model selection process.

Chapter 4

Response

Response to a security incident requires both an accurate analysis of the underlying data and the detection of the incident itself. In network security two kinds of models have been established: The *negative* security model defines via a set of rules, which kind of traffic will not be allowed whereas the *positive* security model fixes the allowed traffic. Therefore, in case of the *negative* security model an intrusion detection system issues an alarm, if one of the rules matches incoming traffic and in case of the *positive* security model the alarm is issued, if the observed traffic deviates from the allowed traffic.

Apart from choosing the type of security model, a network administrator also has to choose a suitable point for deployment in the network infrastructure. Figure 4.1 shows a simple network infrastructure with an internal network, where all clients of the organization reside, and a so-called demilitarized zone (DMZ) where all services are set up, which are accessible from the outside of the organization. The entry to this DMZ is sealed off by a firewall which acts as the central point of entry and departure of network traffic both from the internal network and the DMZ to the outside world. The DMZ acts as another security zone to protect the clients of the organization: the services inside the DMZ have very restricted access to the internal network; thus, if one of these services gets compromised, the attack will not be able to spread inside the internal network.

To further harden specific services one can also deploy application firewalls like the reverse proxy shown in Figure 4.1: this service acts like the protected web server, i.e. all requests to the web server will be first processed by the reverse proxy. This so-called web application firewall (WAF) inspects incoming requests on the application level, since the WAF knows that requests to the web server are encoded in the hypertext transfer protocol (HTTP) and detection of malicious behavior can take place on a fine-grained level.

While an intrusion detection system is merely reporting suspicious network traffic, an intrusion *prevention* system (IPS) reacts in real time to suspicious activities by either *blocking* the traffic or *replacing* parts of the request. Prominent examples are the system `Snort` inline [Roesch, 1999] which can be deployed just like the firewall in Figure 4.1 and the WAF `ModSecurity` which acts like the reverse proxy in the figure. Both solutions work with rule-based detection engines which

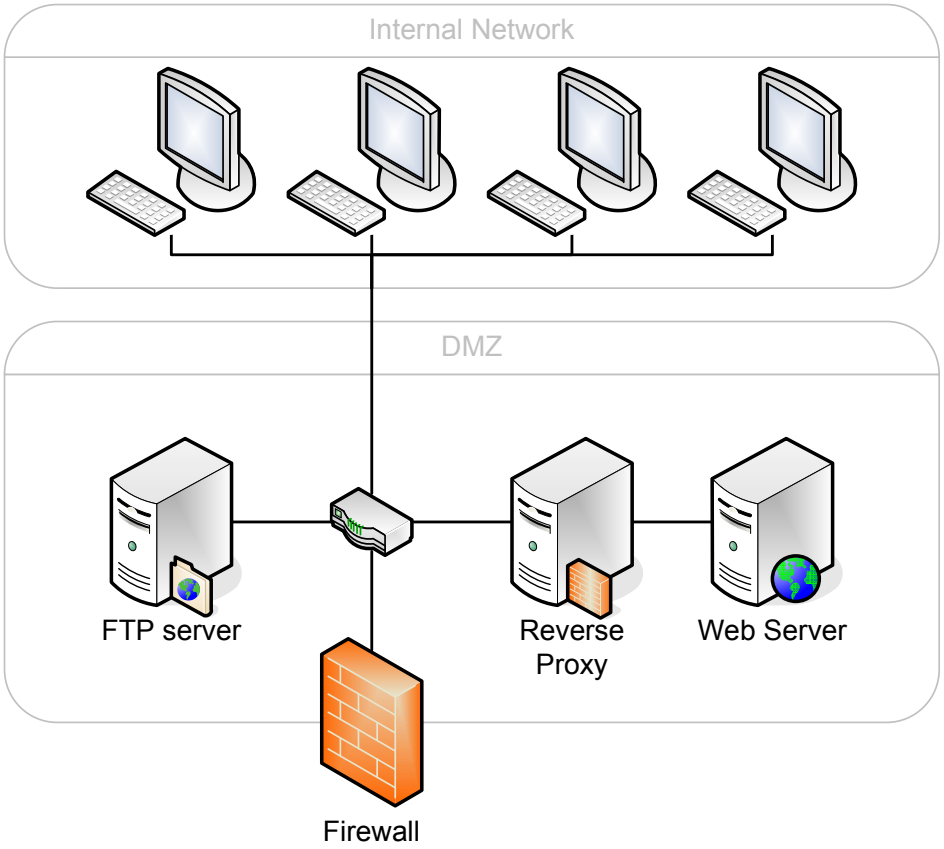


Fig. 4.1: In this network infrastructure the internal network is separated from the so-called demilitarized zone (DMZ). The DMZ contains all servers which are accessible from the outside and is secured by a firewall. The web server is further hardened by an application firewall acting as a reverse proxy to the outside.

define a negative security model, i.e. unless there is not a rule inside the system for a specific exploit, these systems are not able to protect the infrastructure or service at hand.

Approaches based on these negative security models are prone to two major drawbacks: First, they require extensive application and attack knowledge to maintain a reliable rule base. Furthermore, they are only capable of detecting known attacks and leave a system wide open to the so-called *zero-day attacks* for which no patterns are available. A not up-to-date rule base is sufficient for an attack to slip in. Thus, focusing on a positive security model by learning a model of normality helps to circumvent these problems.

In the field of probabilistic methods this problem directly maps to the domain of *unsupervised* learning: Using structural features from Chapter 2 we can learn a model of normality for instance by observing the distribution of the distance to the centroid of the data observed for each specific token. For a new request we can check for each token, whether the obtained distance to the previously calculated centroids are in a statistically sound range according to the observed distributions. Since dif-

ferent tokens of web requests exhibit different behavior, this centroid model might not fit all types of tokens; other approaches like Markov models or even simple lookup tables can often be applied successfully. Given this probabilistic modeling of the token structure we cannot only detect aberrations to normal behavior but we can exploit our models to alter this potentially suspicious request. Using insights derived by the token models we can apply "minimally invasive surgery" to the request and fix just the parts which are suspicious.

In this chapter, we present one instance of such a method which attains the same level of response flexibility as a WAF, namely, the ability not only to drop but also to heal malicious requests, without reliance on known patterns. As already discussed, our approach is based on anomaly detection carried out at the granularity of HTTP request tokens. In contrast to previous applications of anomaly detection to web attack detection, e.g. Kruegel and Vigna [2003], Valeur et al. [2006], Ingham et al. [2007], Cova et al. [2007], Düssel et al. [2008], Song et al. [2009], our method not only detects, but reacts to such attacks.

We have developed a prototype of a reverse proxy called TokDoc which implements the idea of intelligent, token-based substitution coupled with anomaly detection. In our prototype, an HTTP request is parsed into token-value pairs and then compared to learned profiles of normal content of specific tokens. Should some token deviate from a typical profile, it is replaced with an appropriate benign value using token-specific heuristics.

The proposed request healing technique is simple and effective, hence it can efficiently be implemented for deployment in high-speed networks. The main advantage of this so-called *mangling* of requests over simple dropping of requests is that decisions are made in a precise context of specific tokens instead of full requests. This greatly improves detection accuracy, as verified by our experimental comparison with detection at a request level, and makes decisions more fault-tolerant, since the replacement of content with a suitable alternative in certain cases does not harm even if it has been wrongly classified as malicious.

Our system is highly customizable, especially if for some reasons automatic mangling may not be desirable (e.g. for privacy reasons, one may not want to automatically replace user name or password field values). However, in most cases, manual configuration is not necessary, and automatic setup procedures are provided for TokDoc's decision engine. A further advantage of TokDoc is its ability to learn from contaminated data, which enables its deployment with minimal manual effort.

In summary, the contributions of this chapter are:

1. We propose a web application firewall, which decides based on local, anomaly-based models, which parts of a request are anomalous and need to be replaced by benign parts observed in the past.

2. We employ a data-driven setup procedure, which automatically assigns data types to extracted tokens both by structural and statistical features.

3. No "clean", attack-free data set is needed, since both the learning models as well as the tests are robust against "contaminated" data. No additional attack data set is needed for the learning and setup procedure.

The remaining part of this chapter is organized as follows: In Section 4.1 we introduce a methodology for a self-healing web application firewall and present our prototype TokDoc. We evaluate its detection performance and runtime using real HTTP traffic in Section 4.2. Related work is discussed in Section 4.3 and conclusions are given in Section 4.4.

4.1 TokDoc – The Token Doctor

The automatic healing of malicious web requests entails the following two essential tasks: the identification of malicious content and the construction of the replacement content. Unlike regular intrusion detection in which the problem is to decide whether a request is malicious or not, the identification problem is more complex, since one needs to determine not only the presence of malicious content but also its location which is a prerequisite for replacement.

The identification problem can be addressed by making decisions in a refined context of specific parameters of an HTTP request. Such context-dependent detection has been previously used in Kruegel and Vigna [2003] for URI parameters of GET requests and in Düssel et al. [2008] for full HTTP request contents. TokDoc follows roughly the same idea: the requests are parsed into token-value pairs[1], but, instead of combining the scores of token-based models to make decisions at a request level, decisions are made at a token level. Once the content for a token is deemed anomalous, an appropriate replacement is sought.

The diversity of web application traffic content essentially rules out the possibility that a "one-size-fits-all" model of traffic can be learned. Hence we attempt to provide the right anomaly detection algorithm and the appropriate healing action for particular tokens. As a result, the design of TokDoc comprises the following three orthogonal components:

1. *Token Types.* Based on the analysis of real HTTP traffic, we postulate four token types that describe characteristic distributions of token contents.

2. *Anomaly Detectors.* Since malicious content can manifest itself in various features, e.g., unusual length or previously unseen attributes, different anomaly detection algorithms can be used to capture attacks. TokDoc employs a set of anomaly detectors which are automatically coupled with particular tokens in the setup procedure presented in Section 4.1.4.

3. *Healing Actions.* The particular transformation of a token content is denoted as a "healing action". We propose a set of four healing actions that, similar to detectors, depend on the particular token type and are also configurable during the setup procedure.

From the operational point of view, TokDoc can be characterized as a reverse proxy similar in function to a traffic normalizer [Handley et al., 2001]: It intercepts HTTP client requests and, after parsing according to the HTTP protocol

[1] Invalid requests are discarded at this point.

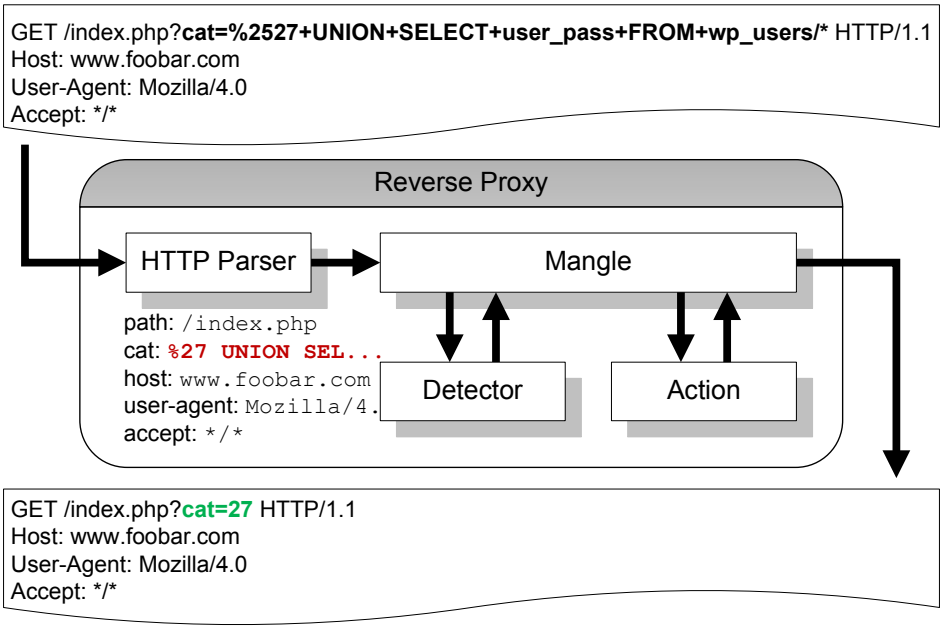


Fig. 4.2: Architecture of TokDoc: The system acts as a reverse proxy, which intercepts each client request. After examination and potential alteration the request is delivered to the production system.

specification and potentially modifying the request content, relays traffic to actual production systems. In contrast to the system proposed by Valeur et al. [2006], TokDoc does not require the target production servers to run at different security levels. The security-related decisions are encapsulated in the proxy itself, which significantly simplifies the design of the network environment to be protected.

The architecture of TokDoc is shown in Figure 4.2. In the example decision presented in this figure, all tokens but `cat` are deemed normal and remain unaltered, whereas the anomalous value of `cat` is flagged for healing. In this example, the value is automatically replaced by previously seen benign data with highest similarity; i.e., the string corresponding to an SQL injection attack is replaced with the benign string 27 due to the occurrence of this string in the attack.

In the following sections we provide a detailed description of the three orthogonal components of TokDoc followed by the presentation of a setup procedure that ties these three components together.

4.1.1 Token Types

Every request received by TokDoc is parsed into syntactical parts according to the HTTP specification and stored as token-value pairs. We treat the URI path, all parameters of GET and POST requests and all header fields as separate tokens followed by respective values. For example, the request in Figure 4.2 is parsed into five tokens corresponding to parameters and headers as denoted inside the reverse proxy. Note, that all request parameters with different names will be presented by different tokens.

The distribution of token content is highly diverse. Some tokens, e.g., `host`, contain only a small number of possible values, sometimes even a constant value. Other tokens such as query and range parameters, contain a wide variety of values, which may even be generated automatically. In order to systematically handle such diversity we classify tokens into four *token types* according to typical properties of information transmitted in HTTP requests:

- *Constants.* In the simplest case the values of a token take the same value, for example the header field `host` when monitoring a particular web host.
- *Enumerations.* A second type of tokens carries data that takes on only a small set of values dependent either on the HTTP protocol itself or on the web application. An example of such token is the `accept-language` header.
- *Machine input.* The third type of tokens comprises machine-generated data, such as session numbers, identifiers and cookies.
- *Human input.* The most complex token type is induced by human input, such as free-text fields, query strings, comments and names. The entered data does not exhibit any semantical structure except for being generated by a natural language.

The characteristic features of different token types have to be taken into account in the choice of anomaly detection algorithms and healing actions.

4.1.2 Anomaly Detectors

Anomaly detection methods have been widely studied for protection of web services [Kruegel and Vigna, 2003, Valeur et al., 2006, Ingham et al., 2007, Cova et al., 2007, Düssel et al., 2008, Song et al., 2009]. However, all previous approaches flag anomalies for full HTTP requests and hence cannot be directly applied for triggering fine-grained actions on individual tokens. In TokDoc we deploy per-token anomaly detection algorithms as proposed by Kruegel and Vigna [2003], however, decision-making remains at the token level.

The choice of an anomaly detection method depends on the token type. For constant and enumeration tokens, a straightforward occurrence check referred to as the LIST detector is a natural choice: if a given value has not been seen in the training data it is deemed anomalous. For the remaining two token types we deploy three different detectors as described below. The decision what detector should be applied to a specific token is automatically made during the setup process presented in Section 4.1.4.

n-gram Centroid Anomaly Detector (NCAD)

n-gram models have been widely used in security applications [Forrest et al., 1996, Rieck and Laskov, 2006, Wang et al., 2006]. In TokDoc, we deploy the embedding technique proposed by Rieck and Laskov [2006], which provides an efficient way for n-gram analysis.

Given the set of all possible n-grams over byte sequences $W = \{0, \ldots, 255\}^n$, we define the embedding function $\phi : W \mapsto \mathbb{R}^{|W|}$ for a token value s as in Section 2.1.2:

$$\phi(s) := (\phi_w(s))_{w \in W} \quad \text{with} \quad \phi_w(s) := \text{occ}_w(s)$$

where $\text{occ}_w(s)$ returns 1 if the n-gram w is contained in s and 0 otherwise. The resulting vector $\phi(s)$ is normalized to one to eliminate length-dependence. The vector space induced by the embedding of n-grams grows exponentially with n, however, its sparseness is linear in the length of sequences. This allows one to efficiently construct and compare embedding vectors $\phi(s)$ for byte sequences as detailed in Rieck and Laskov [2008].

With the embedding function at hand, the Euclidean distance between embedding vectors can be defined as follows:

$$d_e(s,t) = \|\phi(s) - \phi(t)\|_2 = \sqrt{\sum_{w \in W} |\phi_w(s) - \phi_w(t)|^2}.$$

Using this distance, the detection can be performed by computing the distance from a previously learned model μ of normal data:

$$\text{score}_{\text{NCAD}}(s) = \begin{cases} \text{normal}, & \text{if } d_e(\mu, s) \leqslant d_m \\ \text{anomaly}, & \text{otherwise.} \end{cases}$$

The vector μ is constructed from the training data $T = \{t_1, \ldots, t_n\}$ as an arithmetic mean of the respective embedding vectors $\mu = \frac{1}{n} \sum_{i=1}^{n} \phi(t_i)$. The threshold d_m is determined on an independent validation set as described in Section 4.1.4.

Markov Chain Anomaly Detector (MCAD)

Markov chains have previously been used in several security applications [Kruegel and Vigna, 2003, Valeur et al., 2006, Estévez-Tapiador et al., 2004, Song et al., 2009]. We use the 256 possible byte values as states of a Markov chain with 256 possible state transitions each (see Appendix A.3 for details). State transition probabilities can be learned by recording transition frequencies between bytes b_i and b_j in the training data (including an extra start state). The overall size of the transition table is $256^2 + 256$, which is not prohibitively large. Having learned the transition probabilities, we can estimate the probability of a token value s_1^n of length n based on the learned Markov chain:

$$P_{S_1^n}(s_1^n) = P_{S_1}(s_1) \prod_{i=1}^{n-1} P_{S_{t+1}|S_t}(s_{i+1}|s_i)$$

where s_i corresponds to the i-th byte in the token value s_1^n. We do not use length normalization, for instance by applying the geometric mean, because we want a detector which takes both content and length into account. Equipped with the token-specific Markov chain and a threshold p_m the MCAD for a new value s_1^n is defined as follows:

$$\text{score}_{\text{MCAD}}(s_1^n) = \begin{cases} \text{normal}, & \text{if } P_{S_1^n}(s_1^n) \geqslant p_m \\ \text{anomaly}, & \text{otherwise.} \end{cases}$$

Length Anomaly Detector (LAD)

Often it is the length of a token value that is characteristic for an attack. For example, the majority of buffer overflow attacks exhibits long token values. This property is addressed by LAD. This detector is a *fallback solution* for tokens, where an insufficient amount of data renders the learning task involved in training the NCAD and MCAD impossible. Therefore we have to find a solution that can cope with a scarce data situation. Modern robust statistics provides us with powerful tools, which are specialized to deal with noisy data and even small sample sizes. Especially for small sample sizes estimates of the mean and standard deviation as used in the Chebyshev's inequality for instance in Kruegel and Vigna [2003] can be extremely outlier-dependent and a test statistic based on these biased estimates can be too loose.

Hence, instead of using a test based on Chebyshev's inequality, we decide to employ a robust statistic as described in Wilcox [1997]. Given a predefined significance level α_{LAD} we estimate the $1 - \alpha_{\text{LAD}}$ quantile of the length distribution of the train and validation data L, namely $\hat{L}_{1-\alpha_{\text{LAD}}}$. Now we construct a confidence interval for $L_{1-\alpha_{\text{LAD}}}$ by first calculating the bootstrap estimate of the standard error of $\hat{L}_{1-\alpha_{\text{LAD}}}$, namely $\hat{\sigma}$, and determining the parameter c, so that the interval

$$(\hat{L}_{1-\alpha_{\text{LAD}}} - c\hat{\sigma}, \hat{L}_{1-\alpha_{\text{LAD}}} + c\hat{\sigma})$$

has probability coverage of $1-\alpha_{\text{LAD}}$. Finally we choose the upper bound of the confidence interval as a threshold for the LAD detector to allow for future variability. This results in the following decision rule:

$$\text{score}_{\text{LAD}}(s) = \begin{cases} \text{normal}, & \text{if } \text{len}(s) \leqslant \hat{L}_{1-\alpha_{\text{LAD}}} + c\hat{\sigma} \\ \text{anomaly}, & \text{otherwise.} \end{cases}$$

4.1.3 Healing Actions

The fine-grained detection at the token level allows us to devise similarly fine-grained healing actions. Hence our automatic response mechanisms can be less intrusive and more accurate than actions taken at the request level. In particular, TokDoc is equipped with the following healing actions:

- *Dropping of tokens.* The most conservative response to the spotting of an anomalous token value is to remove the token from a request. Notice that this is still a much more benign action than dropping the request itself. We use this action for each token which has an LAD detector.

- *Preventive encoding.* An alternative but still conservative strategy is to encode the anomalous value using HTML entities. This approach makes common web attacks based on cross-site scripting and SQL injection fail, as control and punctuation characters are escaped. This action provides almost no damage to benign requests, as many web applications can resolve additional encoding of content.

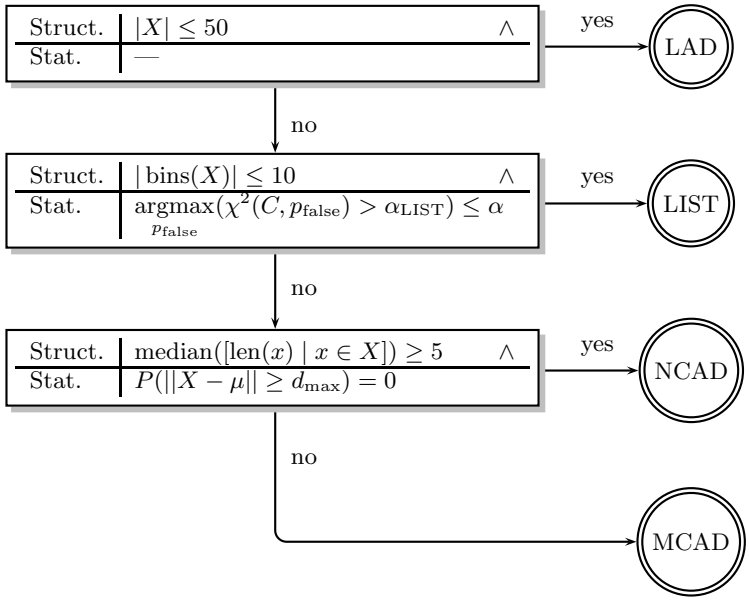


Fig. 4.3: Automatic testing procedure for the setup of TokDoc. After a service-specific split of the training data the testing procedure decides for each token, which detector should be used. By exploiting both structural and statistical features this automatic process is totally data-driven. X denotes the training data for a specific token under test.

- *Replacement with most frequent value.* For constant and enumeration token types, a natural healing action is to replace the value with the most frequent normal value of the token. This is the natural action assigned to a token having the LIST detector.

- *Replacement with nearest value.* The most involved healing action is to replace an anomalous value with its nearest-neighbor from the training set. Such replacement is possible due to the embedding of values in a metric space introduced in Section 4.1.2. This is the default action for both the MCAD and NCAD. Note that as a side-effect, this action can also correct typos in user-input fields.

Clearly, the four healing actions above are tightly coupled with the particular data types of the considered tokens. The precise assignment of healing actions to token types is presented in Section 4.1.4, which also allows the administrator to tighten the proposed default actions for special tokens in need of extra protection like password files and cookies.

4.1.4 Setup of TokDoc

Since the main components of TokDoc are based on learning methods, its setup is dependent on the availability of an initial corpus of normal data for training and

validation. Initially this sufficiently large pool of client requests should be separated according to services (e.g., by virtual hosts and/or different web services) to allow for service-specific learning of models. This data is parsed and used to generate token-specific data pools used in the following phases. The amount of data should be chosen according to traffic volume so that the widest possible range of normal behavior is covered. Note, that the service-specific splitting process of the data could be further improved by the PRISMA framework introduced in Chapter 2 to generate even more focused, state-specific TokDoc instances.

The testing framework depicted in Figure 4.3 determines for each token an automatic and data-driven detector assignment by exploiting both structural and statistical features. Using robust, outlier-resistant statistics, this procedure ensures meaningful decisions even for "dirty", attack-tainted data sets. The collected data is split into two equally sized parts: The training pool is used to learn a model for each token, for which a threshold is estimated using the validation data set. After the semi-automatic assignment of actions and outlier adjustment of thresholds for each token, the TokDoc system is ready for deployment. While the model learning has already been discussed in Section 4.1.2 we now describe in detail the other parts of the setup process.

The data-driven detector assignment is depicted in Figure 4.3. Each step consists of a structural and a statistical test which is carried out for each token in the original data set. Starting with a size test, the procedure assigns the simple LAD detector, if the training data of the currently tested token contains 50 or less samples. The rationale here is that all other detectors need a reasonable amount of data for the estimation of their models. If more than 50 samples are available, the procedure checks, whether the current token is an enumeration. If we observe less than 10 unique values in a token, the procedure tests for statistical evidence by exploiting the well-known χ^2-test. First we define the list $C = [d \in train \mid d \in validate]$, which describes, whether each sample of the validation data set has been observed in the training data set. Then we can define the function $\chi^2(C, p_{\text{false}})$, which returns the p-value of the χ^2-test, whether C could be generated by a binomial variable, which generates "false" with probability p_{false} and "true" with probability $1-p_{\text{false}}$. Now we can determine the maximal p_{false}, that barely supports the acceptance of the hypothesis, that C is generated by $p_{\text{false}}$with a given significance level α_{LIST}:

$$p_{\text{worst-case}} = \underset{p_{\text{false}}}{\text{argmax}}(\chi^2(C, p_{\text{false}}) > \alpha_{\text{LIST}})$$

The value of $p_{\text{worst-case}}$ gives an impression of the possible non-matching occurrences for this token, that might occur in the future or similarly can be interpreted as the upper bound of the confidence interval of the empirical observed p_{false}. Thus we can use this value for thresholding the expected false-positives per token for the LIST data type.

When deciding between NCAD and MCAD, the test procedure first looks at a structural feature, namely the median length of the token. Since the NCAD detector is based on 2-grams, the detector needs at least two characters for calculating a meaningful mean and distances. If the token passes the structural test, the test procedure focuses on a statistical property: Observe that, given the cen-

troid μ, the largest distance from it is bounded by $d_{\max} = \sqrt{\|\mu\|^2 + 1}$ since the data is normalized to a length of one. By using a kernel density estimator on the validation data (see for instance Silverman [1986] for details) we can measure and bound the probability that the maximal distance is ever attained, formally $P(||X - \mu|| \geqslant d_{\max}) = 0$.

Both the NCAD and the MCAD need a threshold for operation. Since the models are focused on a specific token, we can choose a relatively relaxed thresholding policy. We propose to use the maximal distance for NCAD and minimal probability for MCAD, after a semi-automatic outlier adjustment: All values of the validation data set are ordered by the according output of the detector (descending distances to the mean for NCAD and ascending probabilities for MCAD) and the administrator decides, whether the extremal value is a real, user-generated sample or a malicious token value. For example the sorted probabilities in Figure 4.4 clearly show, that the extremal three values are induced by malicious input and therefore the administrator adjusts the threshold to the first user-generated request. During this procedure the administrator additionally can check the quality of the assigned detector and see, whether the chosen model fits the actual data.

In addition he can address privacy and security issues by refining the assigned actions. The administrator can manually adjust, whether a token should be healed or dropped completely. For instance, privacy-related data such as cookies or passwords must not be replaced by its nearest counterpart but instead dropped completely to prevent potential abuse like session or password hijacking.

If the system produces false positives after deployment, these can be tracked down to the token, which caused the false alarm. Thus, the administrator can focus on a specific token and can reconfigure the system according to the incident. In case the website is restructured or new services are deployed, the data model may have to be adjusted accordingly, potentially leading to a retraining of some token models.

4.2 Evaluation

Evaluation of an intrusion prevention system is a multi-faceted task. Since the effectiveness of response actions inherently depend on the accuracy of malicious content identification, we first evaluate the accuracy of TokDoc detectors and compare its overall performance to other state-of-the-art methods. To check for real-time readiness the runtime of TokDoc is assessed and compared to other proxies.

4.2.1 Detection Performance

For the evaluation of detection performance we have collected network traces at two different Internet domains. The first data set (FIRST08) comprises 60 days of traffic with 1,452,122 HTTP requests recorded at the web server of a research institute in 2008. The server provides static content as well as dynamic pages using the content management system `OpenWorx`. The second data set (BLOG09) covers 33 days of traffic with 1,181,941 requests which have been obtained from a domain running various weblogs in 2009. All blogs run on the popular publishing platform

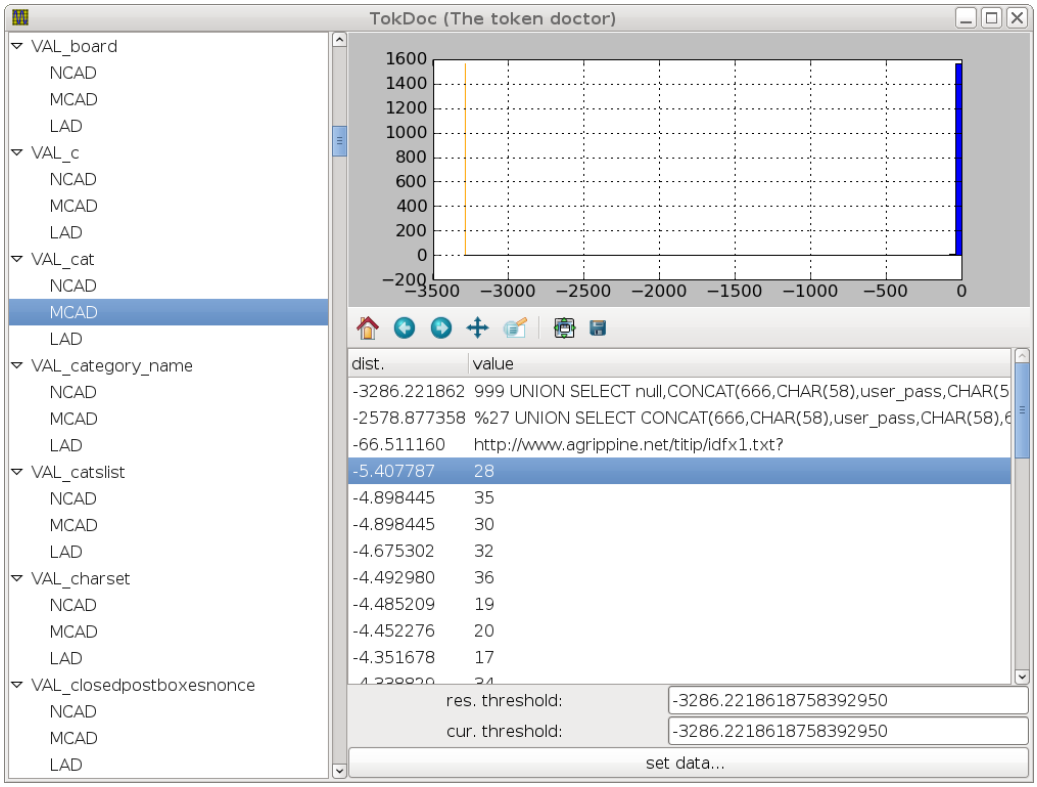


Fig. 4.4: The TokDoc setup console. After the automatic detector assignment the administrator should check the calculated thresholds for outliers.

`WordPress`. For the evaluation, both data sets are split into three equally sized parts for training, validation and testing. Due to the different web applications, the amount of monitored tokens as well as the assignment of anomaly detectors differs between the data sets. The TokDoc configuration used for both data sets is presented in Table 4.1.

In addition to regular network traffic, we have collected network attacks based on 35 exploits obtained from the `Metasploit` framework as well as from common security archives, such as milw0rm, Packet Storm or Bugtraq. Each attack has been executed in a virtual environment and thoroughly adapted to the characteristics of the two data sets. A detailed listing of the considered attacks and exploits is given in Table 4.2. As a result of variations during recording, e.g., usage of different shellcode encoders or SQL statements, the attack pool contains 89 attack instances for FIRST08 and 97 attacks for BLOG09.

Ensemble of Learners and Request Semantics

First of all we want to check, whether all the different detector models are really necessary. For this, we construct special TokDoc instances, which have just the LAD detector instead of both the MCAD and NCAD (referred to as $\mathrm{TD}_{\mathrm{LAD}}$), or just the NCAD (i.e. all MCADs are replaced by NCADs, referred to as $\mathrm{TD}_{\mathrm{NCAD}}$)

	Detectors FIRST08				
Category	LIST	LAD	MCAD	NCAD	Σ
Header	14	14	5	10	43
Parameter	9	3	4	—	16
Path	—	—	1	—	1
Σ	23	17	10	10	60
	Detectors BLOG09				
Category	LIST	LAD	MCAD	NCAD	Σ
Header	22	77	15	17	131
Parameter	14	166	28	7	215
Path	—	—	1	—	1
Σ	36	243	44	24	347

Tab. 4.1: TokDoc configurations used in the experiments. The column category summarizes the tokens into tokens originating from headers, parameters from queries and the path token as introduced in Section 4.1.1.

or MCAD (i.e. all NCADs are replaced by MCADs, referred to as TD_{MCAD}). We evaluate each of these TokDoc instances on both the FIRST08 and BLOG09 data set. Each rejected request is manually checked and labeled as false or true positive, i.e., a rejected request which is originally from the normal traffic and turns out to be a malicious request counts as attack in the normal traffic (ANT). In case of doubt a request is replayed against the original server as follows: First we use the unmodified request and save the reply of the server. This is compared to the reply of the server when we send the request modified by TokDoc. If there is a difference, we count this request as a false positive. In the complete replaying process we could not observe any severe or drastic replies from the servers indicating malformed or even malicious requests. This shows, that the inherent request semantic is not harmed by the actions of TokDoc. The results are summarized in Table 4.3. The first thing to notice is the overall low false-positive rate, which is a direct result of the additional parsing and local decision making of TokDoc. A closer look reveals, that both the TD_{LAD} and TD_{NCAD} suffer from high false-negative rates, while the TD_{MCAD} performs equally good on the FIRST08 data set but falls behind TokDoc on the more involved BLOG09 data. The plain TokDoc with its diversity of models is the only method, which performs nearly identical on both data sets and is also capable of detecting the most true positives in the tainted BLOG09 data. This trend is further confirmed by Table 4.2: All presented detectors are necessary to disarm the used attacks. Note, that the malicious parts of the attacks are spread throughout different tokens rendering the TokDoc approach even more valuable. Additionally, the healing actions employed in TokDoc save roughly 0.0001 of the data from being discarded as false positives on both the BLOG09 and the FIRST08 data set. In summary the results show, that only the full variety of models embodied in TokDoc leads to an overall good performance while keeping the general request semantic intact.

CVE / milworm	Token	Detector
Buffer overflow attacks		
1999-0874	`Path`	MCAD
2001-0241	`Path`	MCAD
2001-0500	∅ (Protocol viol.)	Parser
2002-0392	∅ (Protocol viol.)	Parser
2003-0471	Param. `User`	Normalizer
2003-1192	∅ (Protocol viol.)	Parser
2004-1561	∅ (Protocol viol.)	Parser
2004-1134	∅ (Protocol viol.)	Parser
2005-4734	Param. `url`	Normalizer
2006-1148	∅ (Protocol viol.)	Parser
2006-0992	`Accept-Language`	MCAD
2006-5216	`Path`	MCAD
2006-5478$^{\text{first}}$	`Host`	LIST
2006-5478$^{\text{blog}}$	`Host`	NCAD
Code injection attacks		
2005-0116	Param. `configdir`	Normalizer
2005-0511	Param. `template`	Normalizer
2005-1921	`Body`	MCAD
2005-2847	Param. `f`	Normalizer
2006-1551	`Body`	MCAD
2007-0774	`Path`	MCAD
php_inject	Param. `z`	Normalizer
WordPress attacks		
2004-1584	`Body`	MCAD
2005-1810	Param. `cat`	MCAD
2005-2612	`Cookie`	MCAD
2007-1599	Param. `redirect`	MCAD
7738$^{\text{milwOrm}}$	Param. `thread`	Normalizer
2008-1982	Param. `ss_id`	Normalizer
6842$^{\text{milwOrm}}$	Param. `id`	LAD
2008-5752	Param. `book_id`	Normalizer
2009-0968	Param. `id`	LAD
2009-1030	`Host`	NCAD
Miscellaneous attacks		
httptunnel	∅ (Protocol viol.)	Parser
2004-1373	∅ (Protocol viol.)	Parser
2007-1286	∅ (Protocol viol.)	Parser
xss/sql_injection	Param. `s`	MCAD

Tab. 4.2: Table of CVE numbers for HTTP exploits. Each attack is executed in different variants. We have listed the tokens, in which the attack is located and its detector. In case a token has never been seen before, TokDoc normalizes the request by dropping this token.

Data Set	Detector	FP	#ANT	FN
FIRST08	TokDoc	0.00002	0	0.00000
	TD_{LAD}	0.00000	0	0.02247
	TD_{MCAD}	0.00001	0	0.00000
	TD_{NCAD}	0.00002	0	0.22472
BLOG09	TokDoc	0.00003	212	0.04124
	TD_{LAD}	0.00001	68	0.15464
	TD_{MCAD}	0.00009	186	0.04124
	TD_{NCAD}	0.00003	0	0.22680

Tab. 4.3: Detection performance of several instances of TokDoc (FP = false-positive rate. #ANT = attacks found in normal traffic. FN = false-negative rate).

Data Set	Detector	FP_{TD}	FN_{TD}
FIRST08	TokDoc	0.00002	0.00000
	Markov Chain	0.02005	0.80899
	Anagram	0.00004	0.16854
BLOG09	TokDoc	0.00003	0.04124
	Markov Chain	0.16698	0.18557
	Anagram	1.00000	0.39175

Tab. 4.4: Detection performance of TokDoc and payload-based anomaly detectors. FP_{TD} = false-positive rate of detector when calibrated to the true-positive rate of TokDoc. FN_{TD} = rate of missed regular attacks when detector is calibrated to the false-positive rate of TokDoc.

Comparison to Other Detectors

As a baseline for detection performance, we consider two state-of-the-art anomaly detection techniques using the raw HTTP request payload as input: The *Markov Chain* detector uses a Markov chain as described in Section 4.1.2 over the full content of the requests for anomaly detection. It is learned on the same training data as TokDoc and similarly calibrated using the validation partition. As second baseline, we have implemented a variant of *Anagram* [Wang et al., 2006]. The detector stores n-grams of benign HTTP requests in a Bloom filter and uses the ratio of unknown n-grams in incoming requests as anomaly score. The detector is calibrated on the validation data, and n is fixed to two.

The results of our evaluation are summarized in Table 4.4. For both the FIRST08 and BLOG09 data set we report the FP_{TD}, which equals the false-positive rate of a detector calibrated to the true-positive rate of TokDoc, and FN_{TD}, which is the rate of missed regular attacks, where each detector is calibrated to the false-positive rate of TokDoc. Focusing on the FIRST08 data set we see, that both TokDoc and Anagram yield an acceptable false-positive rate, however Anagram is much more porous: nearly 17% of the attacks are not detected. On the contrary

	Proxy			
Data Set	`Squid`	`ModSec.`	`twisted`	TokDoc
FIRST08	1.387	1.536	2.552	2.768
BLOG09	1.500	1.694	2.430	2.902

Tab. 4.5: Median runtime per request in milliseconds of different proxies for both FIRST08 and BLOG09 data.

TokDoc is capable of detecting all attacks while attaining even a lower false-positive rate than Anagram. The Markov chain is simply overburdened with the FIRST08 and even more with the BLOG09 data set, where its false-positive rate rises to unacceptable 17%. Surprisingly Anagram breaks down on the BLOG09 data set: when calibrated to the true-positive of TokDoc, Anagram flags *all* legitimate requests as anomalous. This is due to the fact that 23% of the attacks have an anomaly score of 0, which is the smallest possible score attainable, therefore tagging all incoming requests as anomalous. But even if we calibrate Anagram to a 23% false-negative rate (roughly 8 times higher than TokDoc in this setup), it yields still a false-positive rate of 0.00038, which is a magnitude higher than TokDoc. These numbers clearly demonstrate the outstanding performance of TokDoc both in terms of false positives and negatives even for hard data sets like the BLOG09 data.

4.2.2 Runtime Performance

To deliver inline intrusion prevention, a system itself has to be reasonably fast, since every client request has to pass the reverse proxy without an intolerable delay. In this part, we subject the TokDoc prototype to a stress test to see whether it can be used in a real-time scenario.

Our prototype is implemented in Python using the `twisted` framework. This framework provides a mature interface to a number of network protocols. By reusing its proxy module and integrating an optimized n-gram C library into Python, we were able to produce a full-fledged prototype of the TokDoc system. We replay the complete testing slice of both the FIRST08 and BLOG09 (approximately 500k requests each) to get a stable estimate of the processing time. As a baseline, we measure the processing time with `Squid` as a proxy. Secondly, we consider the `ModSecurity` web application firewall with a minimal setup of rules to assess, how the additional parsing affects the processing time of a request. Furthermore we use a very simple forwarding proxy application implemented in the `twisted` framework to see, how much the `twisted` framework itself imposes on the processing runtime. Finally, we test TokDoc in the same environment. The median runtime of each setup is presented in Table 4.5.

First we can observe, that the two data sets exhibit different baselines: Generally the FIRST08 data set seems to have a simpler structure compared to the BLOG09 data. Furthermore the highly optimized `Squid` and `ModSecurity` are roughly 1 ms per request faster compared to their Python equivalents. When look-

ing at the inter-application differences, we can observe an increase of 0.1 ms and 0.2 ms from `Squid` to `ModSecurity` and 0.2 ms and 0.5 ms from the `twisted` proxy to TokDoc respectively. This implies, that the additional anomaly detection methods employed in TokDoc just add up to roughly 0.1 ms to 0.3 ms per request. These experiments clearly demonstrate that, while there is still room for improvement in terms of runtime, the anomaly detection methods used in TokDoc are suitable for running in an inline system and that TokDoc even in the current, unoptimized state can already be used as an intrusion prevention system.

4.3 Related Work

The automatic protection of web applications is gaining an increasing attention among security researchers. Conventional IDS such as `Snort` [Roesch, 1999] and `Bro` [Paxson, 1999], which rely on specific attack signatures or predefined attack characteristics, cannot provide adequate and timely protection against dynamically changing web attacks. Anomaly detection techniques based on payload analysis, for example Rieck and Laskov [2006], Wang et al. [2006], Krueger et al. [2008], Perdisci et al. [2009], provide the only possibility for detecting previously unknown attacks. These approaches enable protection of different network services and attain sufficient throughput rates, yet the lack of protocol context in their analysis restricts their use in intrusion prevention to simple dropping or redirection of packets.

First protocol-aware methods for detection of attacks in web traffic using anomaly detection have been proposed by Kruegel and Vigna [2003] and extended in ensuing work [Kruegel et al., 2005, Valeur et al., 2006, Robertson et al., 2006]. The main idea of these methods is the combination of multiple anomaly detectors, such as length checks, byte distributions and hidden Markov models, applied to individual URI parameters. Similarly, finite state automata [Ingham et al., 2007] and multiple Markov chains [Song et al., 2009] have been recently proposed for detection of anomalous HTTP requests. Our approach differs from all these methods in that it detects anomalous content in individual tokens instead of combining token-level anomaly estimates to judge the anomaly of complete requests. Such fine-grained detection enables us to devise novel token healing actions which are much less disruptive than dropping requests.

Another line of research combines network anomaly detection with host monitoring. Anagnostakis et al. [2005] proposed a system in which anomalous requests are executed in a specially instrumented "shadow honeypot" system. The feedback, whether the request actually harms a system, can then be used to update an anomaly detector, similarly to the work of Locasto et al. [2005]. In line with this idea, Vigna et al. [2009] combine SQL attack detection and a reverse proxy to forward requests to web servers, which manage different levels of sensitive information, depending on the anomaly value of the web request. An SQL query anomaly detector on the host decides whether or not the models for the web request anomaly detector should be updated, if the request results in a malicious database query. TokDoc does not require any additional host instrumentation and serves as a transparent proxy, which greatly simplifies its practical deployment.

4.4 Outlook and Conclusion

We have introduced a protocol-aware reverse proxy TokDoc which is capable of deciding at a token level which parts of a request are deemed normal and which anomalous. Several intelligent mangling strategies for anomalous tokens, apart from just dropping them, have been described. Experiments on real-world data sets demonstrate the usefulness of the approach, and runtime measurements show its readiness for inline intrusion prevention.

While the prototype showed good performance, especially in terms of false negatives, we are aware of several extensions that can improve and extend the system. Practical considerations include the integration of TokDoc into `Squid` or the `ModSecurity` platform, which would be a valuable step towards runtime improvement. Application in other domains like database intrusion detection [Bockermann et al., 2009] with its structured query language would be a promising direction. The coupling of the system with a shadow system as proposed in Anagnostakis et al. [2005] and incorporation of a feedback loop in a similar vein to Locasto et al. [2005] in combination with learning techniques is another promising extension. In the case of TokDoc we are able to leverage the knowledge of the underlying protocol to do an efficient preprocessing of the data by parsing the incoming request according to the HTTP grammar. For more complex web applications or web servers which serve a multitude of services there might be a significant overlap of different uses of token types according to the underlying state of the service or the different types of services hosted. Thus, application of methods from Chapter 2 as a further layer to the response mechanism could be beneficial: For each session we would track the state of the service according to the inferred abstract state machine. Anomaly models in sequence are then dependent on this state and an even finer distinction of abnormal behavior could be achieved. This mechanism would also amount to the integration of session-awareness and long-term memory into TokDoc, which could be used for flagging user sessions in which a lot of anomalous tokens have been seen as dangerous. This could be used to reliably close down suspicious connections and even track down attacks distributed over several requests.

Overall, the TokDoc system has proven to be a promising, full-fledged web application firewall in the present state, which is capable of effectively preventing and "healing" a wide range of recent web-based attacks. Its runtime performance makes it readily applicable for protection of modern web applications. Once again we see a layered structure in the processing tool-chain: After the parsing of a request we embed each token in a specifically chosen, localized representation which enables the subsequent check for statistically significant aberrations from the learned model of normality. This modeling approach is even taken a step further in the sense, that we map back our outcome of the probabilistic model into meaningful "healing"-reactions which act as a transformation of information from the probabilistic domain back to the real world. The layered structure of the TokDoc approach permits the stepwise development and successful application of probabilistic methods to a real-world problem.

Chapter 5

Conclusion

In this chapter we summarize the findings of this thesis. We start by reviewing the main parts of the analysis, detection and response architectures which we developed in the preceding chapters. By inspecting the individual components of these frameworks and relating them to the frequentist's view of statistics we gain insights into the preconditions, applications and postconditions of probabilistic methods in network security. We conclude this chapter with an outlook on the implications of our findings, how they relate to "classical" insights of software engineering, and the potential pitfalls and merits of real-world applications.

5.1 From Analysis to Response

Giving a broader view of the results of this thesis in its totality we first have to step back and inspect the individual parts of each developed solution of the security cycle at a more coarse level.

Starting with the analysis framework, we have seen that after a preprocessing of the raw network traffic we are able to represent the data as vectors in a *vector space*. Picking just relevant dimensions of these vectors leads to a testing-based *feature selection* which enables us to shrink down the data. Application of *subspace identification*, for instance via the replicate-aware non-negative matrix factorization or other clustering techniques, leads to a massively reduced representation of the original data. This condensed view of the data allows us to extract an approximation of the abstract state machine by the probabilistic concept of *Markov models*. This model is then reconnected to the actual, non-compressed messages by enumerating all samples in the training pool, aligning them to the extracted states of the abstract state machine, and *inferring* most likely templates and *rules*.

In the detection part we estimate the *model complexity* via the *expected risk* on a hold-out set of data. Leveraging the *convergence* of the expected risk *on subsets* we can transform the performance of a classifier in a reduced, binary trace matrix which records whether the classifier was among the top-performers or not. This binary performance indicator is calculated via *robust testing* on the results of a model on hold-out data, therefore taking advantage of pooled and non-parametric comparison procedures which act with a pre-specified significance level to control

for possible erroneous decisions. This binary trace of each model is then inspected by a *sequential testing* scheme which transforms the individual performance measures on subsets into a global decision whether the model is a significant winner or loser compared to all other models.

The response architecture acts on data in a predefined, structured format. Therefore, we can *parse* the data and find *localized embeddings* which best fit individual parts of the data found in the training pool. After this transformation of the input data into a suitable feature space we can build a *model of normality* from these features which represents the normal behavior found in the observed traffic in a statistical sense. An aberration of this behavior can then be formulated via probabilistic *anomaly detection* techniques based on the individual model for the feature. The concrete model for each feature is chosen based on a *data-driven setup* procedure which based on structural and statistical properties decides which type of model and response strategy fits the feature best.

This high-level description of the solutions presented in this thesis shows that in general we can observe a transformation of the raw data into a condensed feature space which enables a probabilistic modeling of underlying principles. In the next section we discuss the connections of this transformation scheme with the theory of statistical modeling. By ordering the individual steps of each part of the security cycle we can observe that during this transformation procedure the application of probabilistic methods naturally emerges and answers the need for practical information extraction in real-world scenarios.

5.2 Application of Probabilistic Methods

Before we delve deeper in the subject of connections between statistics, probabilistic modeling and transformation processes we have to define what we mean by statistics and probability. Fisher [1922], one of the founders of modern statistics, states in his work *On the mathematical foundations of theoretical statistics*:

> [B]riefly, and in its most concrete form, the object of statistical methods is the reduction of data. [...] This object is accomplished by constructing a hypothetical infinite population, of which the actual data are regarded as constituting a random sample. The law of distribution of this hypothetical population is specified by relatively few parameters, which are sufficient to describe it exhaustively in respect of all qualities under discussion.

In essence, the first step in statistics is to define a sufficient distribution model of the data, which is able to describe the data at hand and also generalizes to new, not yet observed outcomes. Given this distribution model which extends our knowledge of the data beyond the actual pool of data we can start to reason about properties of this hypothetical population. Fisher [1922] defines the term probability as follows:

> When we speak of the *probability* of a certain object fulfilling a certain condition, we imagine all such objects to be divided into two classes,

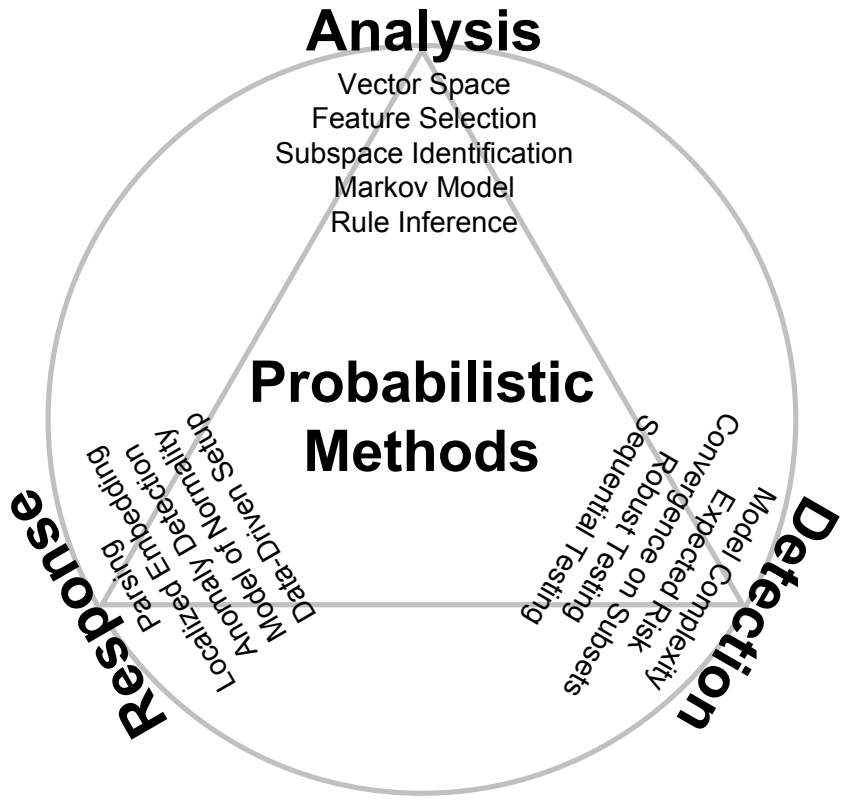


Fig. 5.1: By ordering the individual steps of each part of the security cycle we can observe that each individual step exhibit an increasingly higher proportion of probabilistic methods going from the transformation to the actual modeling of the data.

> according as they do or do not fulfill the condition. [...] It is a parameter which specifies a simple dichotomy in an infinite hypothetical population, and it represents neither more nor less than the frequency ratio which we imagine such a population to exhibit.

We see that probabilistic reasoning about this hypothetical population amounts to the mere observation of frequencies. But in combination with the power of the infinite the frequentist statistician is able to give mathematically sound predictions of the future behavior of the modeled data. Therefore, the preceding description of the data pool via a distribution model offers the modeler a way to extend his knowledge of the previously seen data to future occurrences in a structured and sound manner.

How does this relate to the observations about the transformation steps found in the security cycle? Figure 5.1 shows the individual steps of each framework ordered by their occurrence in the execution. In general, we can observe a transition from the physical preprocessing of the data to a probabilistic one which is the input to the final probabilistic model of the data. For instance, in the analysis part we first preprocess the binary communication data with several preprocessing tools into a vectorial representation. This finite representation subsequently is

	Analysis	Detection	Response
Physical preprocessing	Embedding in vector space	Determine model complexity	Parsing of language
Probabilistic preprocessing	Feature selection	Estimate expected risk	Find localized embeddings
	Identify suitable subspace	Exploit convergence on subsets	Use anomaly detection
Probabilistic modeling	Build Markov model	Robust testing of similar behavior	Learn model of normality
	Infer valid rules	Sequential testing of model performance	Data-driven setup procedure

Tab. 5.1: By a linear ordering of each step a layered structure emerges where lower layers take as input the output from the previous layers. Note that the amount of probabilistic methods involved in the description and modeling of the problem increases from layer to layer.

transformed into a much more condensed version based on statistical tests. This shrunken-down version of the data enables the probabilistic modeling of the data with a low-dimensional matrix factorization or clustering model which is the input to the final Markov model describing the abstract state machine of the service under consideration. Thus, the two-step procedure in classical frequentist statistics is more or less also apparent in the modeling of more complex data: The initial transformation of the physically preprocessed data into a downsized version corresponds to a law of distribution of a hypothetical infinite population of the data which can then be used to gain statistically sound, probabilistic insights into the data. We will see in the next section, how this correspondence can be exploited to efficiently structure probabilistic modeling of complex data for real-world applications.

5.3 Summary and Outlook

The high-level view of each part of the security cycle revealed that before we are able to model and apply probabilistic models we first have to apply a *physical preprocessing* to transform the unstructured, raw data into a suitable form. This amounts in the case of the analysis framework to embedding the data into a vector space, for the design framework in the description of the model complexity via specific meta parameters, and for the response framework in parsing the data according to a specific grammar. Thus, this initial transformation step lays the ground for all subsequent analysis.

Based on the preprocessed data we can proceed by giving a first distributional description of the data. By this *probabilistic preprocessing* we provide the means for higher level reasoning about the data. In the analysis framework we shrink down the number of features based on background knowledge on communication protocols by statistical testing and find a suitable subspace via matrix factorization

or clustering techniques. Exploiting the convergence of the expected risk of a learning model we shrink down the performance of a multitude of models to a simple binary trace matrix which describes the overall performance on the data in the design framework. By defining localized embeddings and using anomaly detection we are able to capture the essentials of the data in a probabilistic fashion for the response framework.

This highly simplified description of the data gives us a solid understanding of the distributional properties of the data and permits the *probabilistic modeling* of our data according to the problem we want to solve: In the analysis framework we model the abstract state machine via a Markov model which incorporates information from the physical and probabilistic preprocessing and facilitates the rule inference for filling the templates. Robust testing procedures and sequential analysis allows us to determine underperforming models in the binary trace matrix with statistical guarantees and even to stop the whole process for an efficient optimal model finding process in the detection framework. Given the possible localized embeddings a data-driven setup procedure automatically decides based on structural and statistical features of the data which model of normality is suitable for specific token types in the response framework.

This layered approach is shown in Table 5.1 with all three frameworks side by side. We can observe that each layer just takes the output of the previous layer as input. Furthermore, the amount of probabilistic methods used in each layer increases from top to bottom as indicated by the gray shading. Thus, this layered approach achieves a high degree of decoupling which simplifies the actual statistical modeling process of the problem domain: By first concentrating on the distributional features of the data we lay ground for the much more sophisticated statistical modeling of the data. It is no coincidence that this resembles the process of good software design. Obviously, modern data is much more structured and complex that one needs a much more decoupled statistical development process extending the initially introduced two-step procedure of first defining the distribution followed by probabilistic reasoning of Fisher [1922].

During the development of the described solutions to the security cycle the best practice of Unix-style development as described in Raymond [2003] has proven to be highly beneficial for the overall development process: By identifying and decoupling the different layers of the statistical modeling process we can develop highly specialized tools which concentrate just on one step of the global modeling process. All these tools can be easily debugged since they are relatively simple and clearly structured. The total modeling process then amounts to an application of a tool chain, where each tool generates the input for the next one. For instance, in the analysis framework we developed for the *physical preprocessing* two tools, which extract both the messages and session information from the binary network traces recorded at a specific site. This is followed by the *probabilistic preprocessing* implemented in the R programming language (see R Core Team [2012]). Both the feature selection and the subspace identification via replicate-aware non-negative matrix factorization can be found in the open source CRAN package PRISMA which is publicly available. Output of this step of the analysis is the input to the PRISMA Python (see Python Core Team [2012]) package which learns the

according *probabilistic model*, i.e., the Markov model, templates and rules based on the input from the previous layers.

We think that not only the software engineering aspect of this layering approach is very fruitful as a design principle for new probabilistic modeling projects. Releasing each part of the processing chain as publicly available, open sourced software packages allow others to benefit from previous work done by other researchers. For instance, by releasing the `CVST` as a CRAN package (see R Core Team [2012]) the fast model selection can be used out of the box in other software projects. As pointed out by Sonnenburg et al. [2007] this not only leads to better software but also faster advancement cycles since the time spent for re-implementing other methods can be used much more beneficially for other aspects in the development of new methods. This is even more the case in the domain of software security due to the high overhead of data preprocessing. Hence, highly modularized and publicly available tools definitely will benefit the overall progress of probabilistic methods in security research.

Real-life application of methods gains increasing attention not only in the machine learning community (see for instance Wagstaff [2012] for a generic discussion and Sculley et al. [2011] for a real-world implementation case study). In computer security the real-life implementation and application is often a sine qua non for a method to get accepted (see for instance Jamshed et al. [2012], De Groef et al. [2012] for some recent, very implementation-heavy work). For probabilistic methods we have seen for instance in Section 2.4.1 that translating the results of the probabilistic model back into the real-world application involves additional, often non-probabilistic adjustments. During the course of this thesis the work covered in Krueger et al. [2008] has been implemented in a research institute network. This real-life application has shown, that the work involved to just deploy the solution in a real-life environment is often considerable. Furthermore, privacy issues involved in handling network data even complicate the whole process of gaining insights into the transition of a research idea into the real world (some of the results can be found in Schuster et al. [2010]). While it is certainly necessary to make this transition into the real world, time and publishing constraints often work against this will. We hope that the presented layered approach together with the plea for more open source releases can act as a first step towards the solution of the pressing problem of translating probabilistic methods into real-life applications for network security.

Bibliography

R. Albright, J. Cox, D. Duling, A. Langville, and C. Meyer. Algorithms, initializations, and convergence for the nonnegative matrix factorization. Technical Report 81706, North Carolina State University, 2006.

D. W. Alling. Closed sequential tests for binomial probabilities. *Biometrika*, 53 (1/2):73–84, 1966.

K. G. Anagnostakis, S. Sidiroglou, P. Akritidis, K. Xinidis, E. Markatos, and A. D. Keromytis. Detecting targeted attacks using shadow honeypots. In *Proceedings of the USENIX Security Symposium*, pages 129–144, 2005.

S. Arlot, A. Celisse, and P. Painleve. A survey of cross-validation procedures for model selection. *Statistics Surveys*, 4:40–79, 2010.

P. Armitage. *Sequential Medical Trials*. Blackwell, 1960.

P. L. Bartlett, P. M. Long, and R. C. Williamson. Fat-shattering and the learnability of real-valued functions. *Journal of Computer and Systems Science*, 52 (3):434–452, 1996.

L. E. Baum and J. A. Eagon. An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology. *Bulletin of the American Mathematical Society*, 73(3):360–363, 1967.

M. A. Beddoe. Network protocol analysis using bioinformatics algorithms. Technical report, McAfee Inc., 2005.

R. E. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.

Y. Bengio. Gradient-based optimization of hyperparameters. *Neural Computation*, 12(8):1889–1900, 2000.

J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. C. N. Pereira,

and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2546–2554, 2011.

D. Berry and B. Fristedt. *Bandit Problems: Sequential Allocation of Experiments*. Chapman & Hall, 1985.

L. Bilge and T. Dumitras. Before we knew it: an empirical study of zero-day attacks in the real world. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 833–844, 2012.

M. Birattari. *Tuning Metaheuristics: A Machine Learning Perspective*. Springer, 2009.

M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 11–18, 2002.

M. Bishop. *Computer Security: Art and Science*. Addison-Wesley, 2002.

J. A. Blackard and D. J. Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and Electronics in Agriculture*, 24(3):131–151, 1999.

C. Bockermann, M. Apel, and M. Meier. Learning sql for database intrusion detection using context-sensitive modelling (extended abstract). In *Proceedings of the 6th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA '09, pages 196–205, 2009.

C. Boutsidis and E. Gallopoulos. Svd based initialization: A head start for nonnegative matrix factorization. *Pattern Recognition*, 41(4):1350–1362, Apr. 2008.

R. Braden. Requirements for Internet Hosts - Communication Layers. RFC 1122 (Standard), Oct. 1989. URL `http://www.ietf.org/rfc/rfc1122.txt`. Updated by RFCs 1349, 4379.

J. K. Bradley and R. Schapire. Filterboost: Regression and classification on large datasets. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 185–192, 2008.

M. L. Braun, J. Buhmann, and K.-R. Müller. On relevant dimensions in kernel feature spaces. *Journal of Machine Learning Research*, 9:1875–1908, 2008.

J. Caballero, H. Yin, and Z. Liang. Polyglot: Automatic extraction of protocol message format using dynamic binary analysis. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CSS)*, 2007.

J. Caballero, P. Poosankam, and C. Kreibich. Dispatcher: Enabling active botnet infiltration using automatic protocol reverse-engineering. In *Proceedings of the 16th ACM conference on Computer and Communications Security (CCS)*, 2009.

N. Cesa-Bianchi and G. Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006.

S. Chien, J. Gratch, and M. Burl. On the efficient allocation of resources for hypothesis evaluation: A statistical approach. *IEEE Transactions Pattern Analysis and Machine Intelligence*, 17(7):652–665, 1995.

S. Chien, A. Stechert, and D. Mutz. Efficient heuristic hypothesis ranking. *Journal of Artificial Intelligence Research*, 10(1):375–397, 1999.

W. G. Cochran. The comparison of percentages in matched samples. *Biometrika*, 37(3-4):256–266, 1950.

P. Comparetti and G. Wondracek. Prospex: Protocol specification extraction. In *Proceedings of the 30th IEEE Symposium on Security and Privacy*, 2009.

M. Cova, D. Balzarotti, V. Felmetsger, and G. Vigna. Swaddler: An approach for the anomaly-based detection of state violations in web applications. In *Recent Advances in Intrusion Detection (RAID)*, pages 63–86, September 2007.

W. Cui and J. Kannan. Discoverer: Automatic protocol reverse engineering from network traces. In *Proceedings of the 16th USENIX Security Symposium*, 2007.

W. Cui, V. Paxson, N. C. Weaver, and R. H. Katz. Protocol-independent adaptive replay of application dialog. In *Proceedings of the 13th Network and Distributed System Security Symposium (NDSS)*, 2006.

W. Cui, M. Peinado, K. Chen, and H. Wang. Tupni: Automatic reverse engineering of input formats. In *Proceedings of the 15th ACM conference on Computer and Communications Security (CCS)*, 2008.

A. Dainotti, A. King, K. Claffy, F. Papale, and A. Pescapè. Analysis of a "/0" Stealth Scan from a Botnet. In *Internet Measurement Conference (IMC)*, 2012.

W. De Groef, D. Devriese, N. Nikiforakis, and F. Piessens. Flowfox: a web browser with flexible and precise information flow control. In *Proceedings of the 2012 ACM conference on Computer and communications security*, CCS '12, pages 748–759, 2012.

D. E. Denning. An intrusion-detection model. *IEEE Transanctions on Software Engineering*, 13(2):222–232, 1987.

L. Devroy, L. Györfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition*. Springer, 1996.

C. H. Q. Ding, T. Li, and W. Peng. On the equivalence between non-negative matrix factorization and probabilistic latent semantic indexing. *Computational Statistics & Data Analysis*, 52(8):3913–3927, 2008.

P. Domingos and G. Hulten. A general method for scaling up machine learning algorithms and its application to clustering. In *Proceedings of the 18th International Conference on Machine Learning*, pages 106–113, 2001.

D. L. Donoho and J. M. Johnstone. Ideal spatial adaptation by wavelet shrinkage. *Biometrika*, 81(3):425–455, 1994.

P. Düssel, C. Gehl, P. Laskov, and K. Rieck. Incorporation of application layer protocol syntax into anomaly detection. In *Proceedings of International Conference on Information Systems Security (ICISS)*, pages 188–202, 2008.

J. M. Estévez-Tapiador, P. García-Teodoro, and J. E. Díaz-Verdejo. Measuring normality in http traffic for anomaly-based intrusion detection. *Computer Networks*, 45(2):175–193, 2004.

T. Evgeniou and M. Pontil. On the V gamma dimension for regression in reproducing kernel hilbert spaces. In O. Watanabe and T. Yokomori, editors, *Algorithmic Learning Theory*, pages 106–117, 1999.

F-Secure. Mobile threat report. Q2 2012, 2012. URL `http://www.f-secure.com/weblog/archives/MobileThreatReport_Q2_2012.pdf`.

R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. URL `http://www.ietf.org/rfc/rfc2616.txt`. Updated by RFC 2817.

R. A. Fisher. On the mathematical foundation of theoretical statistics. *Philosophical Transactions of the Royal Society A*, 222:309–368, 1922.

S. Forrest, S. Hofmeyr, A. Somayaji, and T. Longstaff. A sense of self for UNIX processes. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 120–128, 1996.

A. M. Fraser. *Hidden Markov Models and Dynamical Systems*. Society for Industrial and Applied Mathematics, 2008.

M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200): 675–701, 1937.

S. Geisser. The predictive sample reuse method with applications. *Journal of the American Statistical Association*, 70(350):320–328, 1975.

D. Gollmann. *Computer Security*. Wiley, 2002.

M. Handley, V. Paxson, and C. Kreibich. Network intrusion detection: Evasion, traffic normalization and end-to-end protocol semantics. In *Proceedings of the USENIX Security Symposium*, 2001.

D. A. Harville. *Matrix Algebra from a Statistican's Perspective*. Springer, 1997.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009.

N. A. Heckert and J. J. Filliben. *NIST Handbook 148: DATAPLOT Reference Manual, Volume I: Commands*. National Institute of Standards and Technology Handbook Series, 2003.

V. Heidrich-Meisner and C. Igel. Hoeffding and Bernstein races for selecting policies in evolutionary direct policy search. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 401–408, 2009.

M. Heiler and C. Schnörr. Learning sparse representations by non-negative matrix factorization and sequential cone programming. *Journal of Machine Learning Research*, 7:1385–1407, 2006.

P. Hethmon. Extensions to FTP. RFC 3659 (Proposed Standard), Mar. 2007. URL `http://www.ietf.org/rfc/rfc3659.txt`.

P. Hethmon and R. Elz. Feature negotiation mechanism for the File Transfer Protocol. RFC 2389 (Proposed Standard), Aug. 1998. URL `http://www.ietf.org/rfc/rfc2389.txt`.

T. Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '99, pages 50–57, 1999.

P. Holland. Weighted ridge regression: Combining ridge and robust regression methods. Technical Report 11, National Bureau of Economic Research, 1973.

S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70, 1979.

P. O. Hoyer. Non-negative matrix factorization with sparseness constraints. *Journal of Machine Learning Research*, 5:1457–1469, 2004.

K. L. Ingham, A. Somayaji, J. Burge, and S. Forrest. Learning DFA representations of HTTP for protecting web applications. *Computer Networks*, 51(5):1239–1255, 2007.

G. Jacob, R. Hund, C. Kruegel, and T. Holz. Jackstraws: Picking command and control connections from bot traffic. *Proceedings of the 20th USENIX Security Symposium*, 2011.

M. A. Jamshed, J. Lee, S. Moon, I. Yun, D. Kim, S. Lee, Y. Yi, and K. Park. Kargus: a highly-scalable software-based intrusion detection system. In *Proceedings of the 2012 ACM conference on Computer and communications security*, CCS '12, pages 317–328, 2012.

I. T. Jolliffe. *Principal Component Analysis*. Springer, 1986.

S. S. Keerthi, V. Sindhwani, and O. Chapelle. An efficient method for gradient-based adaptation of hyperparameters in svm models. In *Advances in Neural Information Processing Systems 19*, pages 673–680, 2006.

R. Kohavi and G. H. John. Automatic parameter selection by minimizing estimated error. In *Proceedings of the 12th International Conference on Machine Learning*, pages 304–312, 1995.

U. Krengel. *Einführung in die Wahrscheinlichkeitstheorie und Statistik*. Vieweg, 2002.

C. Kruegel and G. Vigna. Anomaly detection of web-based attacks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, pages 251–261, 2003.

C. Kruegel, G. Vigna, and W. Robertson. A multi-model approach to the detection of web-based attacks. *Computer Networks*, 48(5):717–738, 2005.

T. Krueger and K. Rieck. Intelligent defense against malicious javascript code. *PIK - Praxis der Informationsverarbeitung und Kommunikation*, 35(1):54–60, 2012.

T. Krueger, C. Gehl, K. Rieck, and P. Laskov. An architecture for inline anomaly detection. In *Proceedings of the European Conference on Computer Network Defense (EC2ND)*, pages 11–18, 2008.

T. Krueger, C. Gehl, K. Rieck, and P. Laskov. TokDoc: A self-healing web application firewall. In *Proceedings of the 25th ACM Symposium on Applied Computing (SAC)*, pages 1846–1853, March 2010.

T. Krueger, N. Krämer, and K. Rieck. ASAP: automatic semantics-aware analysis of network payloads. *Proceedings of the ECML/PKDD Conference on Privacy and Security Issues in Data Mining and Machine Learning*, 2011.

T. Krueger, H. Gascon, N. Krämer, and K. Rieck. Learning stateful models for network honeypots. In *Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence*, AISEC '12, 2012a.

T. Krueger, D. Panknin, and M. Braun. Fast cross-validation via sequential testing. *Computing Research Repository*, abs/1206.2248, 2012b. URL `http://arxiv.org/abs/1206.2248`.

D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791, 1999.

C. Leita and M. Dacier. Automatic handling of protocol dependencies and reaction to 0-day attacks with ScriptGen based honeypots. In *Proceedings of the 9th international conference on Recent Advances in Intrusion Detection (RAID)*, 2006.

C. Leita and K. Mermoud. ScriptGen: An automated script generation tool for honeyd. In *Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC)*, 2005.

Z. Lin, X. Jiang, and D. Xu. Automatic protocol format reverse engineering through context-aware monitored execution. In *Proceedings of the 15th Network and Distributed System Security Symposium (NDSS)*, 2008.

M. E. Locasto, K. Wang, A. D. Keromytis, and S. J. Stolfo. Flips: Hybrid adaptive intrusion prevention. In *Recent Advances in Intrusion Detection (RAID)*, pages 82–101, 2005.

P. Maass and M. Rajagopalan. Does cybercrime really cost $1 trillion? ProPublica, Aug. 1, 2012, 2012. URL `https://www.propublica.org/article/does-cybercrime-really-cost-1-trillion/`.

D. Mackenzie and G. Pottinger. Mathematics, technology, and trust: Formal verification, computer security, and the U.S. military. *IEEE Annals of the History of Computing*, 19(3):41–59, 1997.

D. Mankins, D. Franklin, and A. Owen. Directory oriented FTP commands. RFC 775, Dec. 1980. URL `http://www.ietf.org/rfc/rfc775.txt`.

O. Maron and A. W. Moore. Hoeffding races: Accelerating model selection search for classification and function approximation. In *Advances in Neural Information Processing Systems 6*, pages 59–66, 1994.

O. Maron and A. W. Moore. The racing algorithm: Model selection for lazy learners. *Artificial Intelligence Review*, 11(1-5):193–225, 1997.

J. McHugh. Sets, bags, and rock and roll. In *Computer Security – ESORICS 2004*, volume 3193 of *Lecture Notes in Computer Science*, pages 407–422. Springer, 2004.

C. K. McPherson and P. Armitage. Repeated significance tests on accumulating data when the null hypothesis is not true. *Journal of the Royal Statistical Society. Series A*, 134(1):15–25, 1971.

MDL. Malware domain list, 2012. URL `http://www.malwaredomainlist.com/`.

M. Meier. *Intrusion Detection effektiv! Modellierung und Analyse von Angriffsmustern*. Springer, 2007.

V. Mnih, C. Szepesvári, and J.-Y. Audibert. Empirical bernstein stopping. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 672–679, 2008.

M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations Of Machine Learning*. MIT Press, 2012.

E. F. Moore. Gedanken-experiments on sequential machines. *Automata Studies*, 34:129–153, 1956.

F. Mosteller and J. W. Tukey. Data analysis, including statistics. In G. Lindzey and E. Aronson, editors, *Handbook of Social Psychology*, volume 2, pages 80–203. Addison-Wesley, 1968.

K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 12 (2):181–201, 2001.

J. Newsome, D. Brumley, and J. Franklin. Replayer automatic protocol replay by binary analysis. In *Proceedings of the 13th ACM conference on Computer and Communications Security (CCS)*, 2006.

Norton. 2012 norton cybercrime report. Symantec Corporation, 2012. URL `http://now-static.norton.com/now/en/pu/images/Promotions/2012/cybercrimeReport/2012_Norton_Cybercrime_Report_Master_FINAL_050912.pdf`.

OSVDB. The open source vulnerability database, 2012. URL `http://osvdb.org/`.

P. Paatero and U. Tapper. Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics*, 5(2):111–126, 1994.

R. Pang and V. Paxson. A high-level programming environment for packet trace anonymization and transformation. *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2003.

K. D. Patil. Cochran's Q test: Exact distribution. *Journal of the American Statistical Association*, 70(349):186–189, 1975.

V. Paxson. Bro: A system for detecting network intruders in real-time. *Computer Networks*, 31(23–24):2435–2466, Dec. 1999.

R. Pelossof and M. Jones. Curtailed online boosting. Technical report, Columbia University, 2009.

R. Pelossof and Z. Ying. The attentive perceptron. *Computing Research Repository*, abs/1009.5972, 2010.

R. Pelossof and Z. Ying. Rapid learning with stochastic focus of attention. *Computing Research Repository*, abs/1105.0382, 2011.

R. Perdisci, D. Ariu, P. Fogla, G. Giacinto, and W. Lee. McPAD: A multiple classifier system for accurate payload-based anomaly detection. *Computer Networks*, 53(6):864–881, 2009.

J. Postel and J. Reynolds. File Transfer Protocol. RFC 959 (Standard), Oct. 1985. URL `http://www.ietf.org/rfc/rfc959.txt`. Updated by RFCs 2228, 2640, 2773, 3659.

Python Core Team. *Python Programming Language*. Python Software Foundation, 2012. URL `http://www.python.org/`.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2012. URL `http://www.R-project.org`.

G. Rätsch, T. Onoda, and K.-R. Müller. Soft margins for AdaBoost. *Machine Learning*, 42(3):287–320, 2001.

E. S. Raymond. *The Art of UNIX Programming*. Addison-Wesley, 2003.

K. Rieck. Computer security and machine learning: Worst enemies or best friends? *DIMVA Workshop on Systems Security (SYSSEC)*, 2011.

K. Rieck and P. Laskov. Detecting unknown network attacks using language models. In *Detection of Intrusions and Malware, and Vulnerability Assessment, Proceedings of 3rd DIMVA Conference*, pages 74–90, 2006.

K. Rieck and P. Laskov. Linear-time computation of similarity measures for sequential data. *Journal of Machine Learning Research*, 9:23–48, 2008.

K. Rieck, T. Krueger, and A. Dewald. Cujo: efficient detection and prevention of drive-by-download attacks. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 31–39, 2010.

W. Robertson, G. Vigna, C. Kruegel, and R. A. Kemmerer. Using generalization and characterization techniques in the anomaly-based detection of web attacks. In *Proceedings of Network and Distributed System Security Symposium (NDSS)*, 2006.

M. Roesch. Snort: Lightweight intrusion detection for networks. In *Proceedings of USENIX Large Installation System Administration Conference LISA*, pages 229–238, 1999.

V. Roth. Probabilistic discriminative kernel classifiers for multi-class problems. In *Proceedings of the 23rd DAGM-Symposium on Pattern Recognition*, pages 246–253, 2001.

S. Schmerl, H. Koenig, U. Flegel, M. Meier, and R. Rietz. Systematic signature engineering by re-use of snort signatures. In *Proceedings of the 2008 Annual Computer Security Applications Conference*, ACSAC '08, pages 23–32, 2008.

R. Schmidt. Multiple emitter location and signal parameter estimation. *IEEE Transactions on Antennas and Propagation*, 34(3):276 – 280, 1986.

B. Schneier. *Secets & Lies*. Wiley, 2004.

B. Schneier. *Liars & Outliers*. Wiley, 2012.

B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319, 1998.

B. Schölkopf, A. J. Smola, R. C. Williamson, and P. L. Bartlett. New support vector algorithms. *Neural Computation*, 12(5):1207–1245, 2000.

I. Schuster, T. Krueger, C. Gehl, K. Rieck, and P. Laskov. Fips: First intrusion prevention system. Technical Report 1, Fraunhofer FIRST, 2010. URL `http://publica.fraunhofer.de/documents/N-148519.html`.

D. Sculley, M. E. Otey, M. Pohl, B. Spitznagel, J. Hainsworth, and Y. Zhou. Detecting adversarial advertisements in the wild. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '11, pages 274–282, 2011.

B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall, 1986.

S. Smale and D.-X. Zhou. Estimating the approximation error in learning theory. *Analysis and Applications*, 1(1):1–25, 2003.

P. Smyth. Exploring text and social network data with probabilistic models. Talk held at TU Berlin, June 2012.

J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In P. Bartlett, F. C. N. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2960–2968, 2012.

R. Sommer and V. Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, SP '10, pages 305–316, 2010.

Y. Song, A. D. Keromytis, and S. J. Stolfo. Spectrogram: A mixture-of-markov-chains model for anomaly detection in web traffic. In *Proceedings of Network and Distributed System Security Symposium (NDSS)*, 2009.

S. Sonnenburg, M. L. Braun, C. S. Ong, S. Bengio, L. Bottou, G. Holmes, Y. LeCun, K.-R. Müller, F. Pereira, C. E. Rasmussen, G. Rätsch, B. Schölkopf, A. J. Smola, P. Vincent, J. Weston, and R. C. Williamson. The need for open source software in machine learning. *Journal of Machine Learning Research*, 8:2443–2466, 2007.

C. C. Spicer. Some new closed sequential designs for clinical trials. *Biometrics*, 18 (2):203–211, 1962.

A. Stanski. *Konstruktives Probabilistisches Lernen*. PhD thesis, Technische Universität Berlin, 2012.

A. Stanski and O. Hellwich. A projection and density estimation method for knowledge discovery. *PLoS ONE*, 7(10):e44495, 2012.

I. Steinwart and C. Scovel. Fast rates for support vector machines using Gaussian kernels. *Annals of Statistics*, 35:575–607, 2007.

M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society. Series B*, 36(2):111–147, 1974.

Symantec. Internet security threat report. Volume 17, Symantec Corporation, 2012.

M. W. Tate and S. M. Brown. Note on the Cochran Q test. *Journal of the American Statistical Association*, 65(329):155–160, 1970.

T. Taylor, D. Paterson, J. Glanfield, C. Gates, S. Brooks, and J. McHugh. Flovis: Flow visualization system. In *Conference For Homeland Security, 2009. CATCH '09. Cybersecurity Applications Technology*, pages 186 –198, 2009.

C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-weka: Automated selection and hyper-parameter optimization of classification algorithms. *Computing Research Repository*, abs/1208.3719, 2012.

F. Valeur, G. Vigna, C. Kruegel, and E. Kirda. An anomaly-driven reverse proxy for web applications. In *Proceedings of the 2006 ACM symposium on Applied computing*, pages 361–368, 2006.

V. Vapnik. *Statistical Learning Theory*. Wiley, 1998.

G. Vigna, F. Valeur, D. Balzarotti, W. Robertson, C. Kruegel, and E. Kirda. Reducing errors in the anomaly-based detection of web-based attacks through the combined analysis of web requests and SQL queries. *Journal of Computer Security*, 17(3):305–329, 2009.

K. L. Wagstaff. Machine Learning that Matters. In *Proceedings of the 29th International Conference on Machine Learning*, 2012.

A. Wald. *Sequential Analysis*. Wiley, 1947.

A. Wald and J. Wolfowitz. Optimum character of the sequential probability ratio test. *Annals of Mathematical Statistics*, 19(3):326–339, 1948.

K. Wang and S. Stolfo. Anomalous payload-based network intrusion detection. In *Recent Advances in Intrusion Detection (RAID)*, pages 203–222, 2004.

K. Wang, G. Cretu, and S. Stolfo. Anomalous payload-based worm detection and signature generation. In *Recent Advances in Intrusion Detection (RAID)*, 2005.

K. Wang, J. Parekh, and S. Stolfo. Anagram: A content anomaly detector resistant to mimicry attack. In *Recent Advances in Intrusion Detection (RAID)*, pages 226–248, 2006.

Z. Wang, X. Jiang, W. Cui, and X. Wang. ReFormat: Automatic reverse engineering of encrypted messages. In *European Symposium on Research in Computer Security (ESORICS)*, 2009.

G. B. Wetherill and K. D. Glazebrook. *Sequential Methods in Statistics*. Chapman & Hall, 1986.

R. R. Wilcox. *Introduction to Robust Estimation and Hypothesis Testing*. Academic Press, 1997.

G. Wondracek and P. Comparetti. Automatic network protocol analysis. In *Proceedings of the 15th Network and Distributed System Security Symposium (NDSS)*, 2008.

Appendix A

Definitions

A.1 Geometrical Concepts in a Vector space

In this section we follow the treatment of Harville [1997] and introduce some general geometrical concepts in vector spaces which are used throughout this thesis. Given a vector space we define the 2-norm of a vector $x = (x_1, x_2, \ldots, x_f)^\top$ as follows:

$$\|x\| := \sqrt{x_1^2 + x_2^2 + \cdots + x_f^2} =: \sqrt{x \bullet x} = \sqrt{x^\top x}.$$

The usual Euclidean distance d_e between two vectors x and y can then be defined as follows:

$$d_e(x, y) := \|x - y\| = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_f - y_f)^2} =: \sqrt{x \bullet y}.$$

With the norm and the law of cosines we get for two vectors x and y the following motivation for an angle (see Figure A.1):

$$\begin{aligned} & \|x - y\|^2 = \|x\|^2 + \|y\|^2 - 2\|x\|\|y\| \cos(\theta) \\ \Leftrightarrow \quad & -2x \bullet y = -2\|x\|\|y\| \cos(\theta) \\ \Leftrightarrow \quad & \cos(\theta) = \frac{x \bullet y}{\|x\|\|y\|}. \end{aligned}$$

If we define the *dot* product for some matrices A, B as follows

$$A \bullet B := \operatorname{trace}(A^\top B) = \operatorname{trace}(B^\top A),$$

where trace calculates the sum of the diagonal entries of the matrix, then all above definitions extend naturally to matrices. For instance using the *Frobenius* norm of a matrix A, i.e. $\|A\| = \sqrt{A \bullet A}$, all the above definitions for distance and angles extend naturally to the domain of matrices.

A.2 Probabilities

In this section we give a quick introduction to the concept of probability theory based on the treatment of Krengel [2002]. First we define a *probability space*

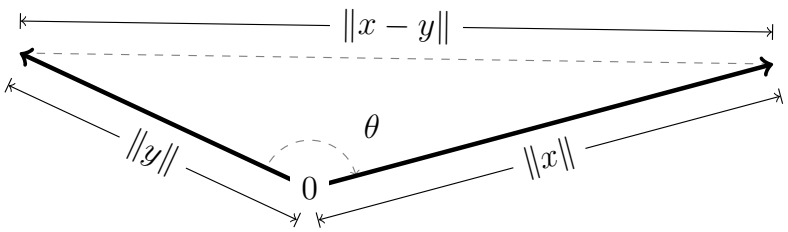


Fig. A.1: Norms and angles in $\mathbb{R}^2$.

(Ω, F, P), where Ω denotes the possible outcomes, F the σ-algebra on it defining the possible events and $P : F \mapsto [0, 1]$ is a *probability measure* satisfying the following axioms:

4. $P(\Omega) = 1$
5. $P(A) \geqslant 0 \quad$ for all $A \in F$
6. $P(\bigcup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} P(A_i) \quad$ for all $A_i \in F$ with $A_i \cap A_j = \varnothing, i \neq j$

We call $P(A), A \in F$ the probability of event A. For two events $A, B \in F$ we define the *conditional probability* as follows:

$$P(A|B) := \frac{P(A \cap B)}{P(B)}.$$

We call two events $A, B \in F$ *independent*, if

$$P(A \cap B) = P(A)P(B).$$

A *random variable* is a function $X : \Omega \mapsto \mathcal{X}$ mapping possible outcomes to a set $\mathcal{X}$. With

$$P_X(A) := P(X^{-1}(A)) \quad A \in \mathcal{X}, X^{-1} \text{ the inverse map}$$

we can define a probability measure for X. We write $X \sim P_X$ to denote the fact, that X is distributed according to the probability measure P_X. We call n random variables $X_i : \Omega \mapsto \mathcal{X}_i$ independent, if for all results $A_i \subset \mathcal{X}_i$ the corresponding events $X_i^{-1}(A_i)$ are independent. We define the *mean* and the *variance* of a random variable X as follows:

$$E[X] := \sum_{\omega \in \Omega} X(\omega)P(\omega) \quad \text{and} \quad Var[X] := E[(X - E[X])^2].$$

For n random variables $X_i : \Omega \mapsto \mathcal{X}_i$ we write for the *random vector* $X := (X_1, X_2, \ldots, X_n) : \Omega \mapsto (\mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_n)$ and sets $x_i \subset \mathcal{X}_i$:

$$P_X(X_1 \in x_1, X_2 \in x_2, \ldots, X_n \in x_n) := P(\{\omega \in \Omega : X_i(\omega) \in x_i, i \in 1, \ldots, n\}).$$

We exemplify these concepts with the help of a *binomial* variable B. Suppose we have n independent experiments, where we can have 1 or 0 as outcome with

probability π and $1-\pi$, respectively. Then, $\Omega = \{0,1\}^n$, and $P(\omega) = \pi^k(1-\pi)^{n-k}$ with k the number of ones in ω. If we define by

$$E_k := \{\omega \in \Omega : \sum_{i=1}^{n} \omega_i = k\}$$

the event that we observe k ones, then

$$P(E_k) = b_{n,\pi}(k) := \binom{n}{k} \pi^k (1-\pi)^{n-k}.$$

Therefore, the random variable $B : \Omega \mapsto \mathbb{N}$ with $B(\omega) = \sum_{i=1}^{n} \omega_i$ follows a binomial distribution, i.e. $B \sim b_{n,\pi}$. If we describe B as the sum of a series of n experiments B_i we can write:

$$E[B] = E[\sum_{i=1}^{n} B_i] = \sum_{i=1}^{n} E[B_i] = \sum_{i=1}^{n} 1 \cdot \pi + 0 \cdot (1-\pi) = n \cdot \pi.$$

Similar, using the independence of the individual experiments B_i we can calculate the variance of B as follows:

$$Var[B] = Var[\sum_{i=1}^{n} B_i] = \sum_{i=1}^{n} Var[B_i] = \sum_{i=1}^{n} E[B_i^2] - E[B_i]^2 = n(\pi - \pi^2)$$

A.3 Markov Models

In this section we give some background to Markov models following the treatment of Fraser [2008]. An indexed sequence of random variables

$$S_1^T := [S_1, S_2, \ldots, S_{T-1}, S_T]$$

is a *Markov chain* if it fulfills the *Markov Assumption*:

$$\forall t \in 1, \ldots, T : P_{S_t|S_1,S_2,\ldots,S_{t-2},S_{t-1}} = P_{S_t|S_{t-1}}.$$

In case of a discrete Markov model the S_i represent the internal states of the system which determine its behavior. We can estimate probabilities of concrete state sequences s_1^T as follows:

$$\begin{aligned} P_{S_1^T}(s_1^T) &= P_{S_1}(s_1) \prod_{t=2}^{T} P_{S_t|S_1^{t-1}}(s_t|s_1^{t-1}) \\ &= P_{S_1}(s_1) \prod_{t=2}^{T} P_{S_t|S_{t-1}}(s_t|s_{t-1}) \end{aligned}$$

For time-homogeneous, i.e. stationary Markov chains we have

$$P_{S_{t+1}|S_t}(s_{t+1}|s_t) = P_{S_t|S_{t-1}}(s_t|s_{t-1})$$

for all t, therefore it is sufficient to estimate the probabilities for the initial state $P(S_1)$ and the state transition matrix $P_{S_t|S_{t-1}}(s_t|s_{t-1})$ for all concrete outcomes of the state space to define the Markov model.

	H_0 is true	H_1 is true
Accept H_0	Right decision	Type II Error (β)
Reject H_0	Type I Error (α)	Right decision

Tab. A.1: Type I and type II error in statistical testing.

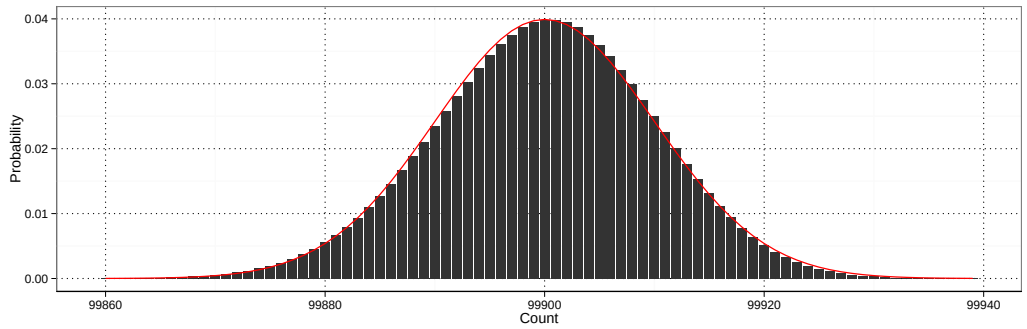


Fig. A.2: Distribution of a feature count $F \sim b_{n=100,000,\pi=0.999}$ overlayed by the normal distribution approximation in red.

A.4 Statistical Testing

In statistical testing we want to decide, whether we can accept or reject a hypothesis (H_0) in favor to an alternative (H_1). For this we have to make some distributional assumptions to backup our arguments in a statistical sense. Then, given a predefined significance level (normally $\alpha \in \{0.01, 0.05, 0.1\}$) we calculate a test statistic given our data and can determine a p-value which describes the probability to observe this or an even more extreme statistic given the hypothesis H_0. If this p-value is smaller or equal to our significance level α we reject the hypothesis H_0 and accept the alternative hypothesis H_1. When testing hypotheses several things can go wrong as summarized in the Table A.1 Basically, we control the type I error by the *significance level* α. The type II error can be used to describe the *power* $(1 - \beta)$ of the test, i.e. the probability of correctly rejecting H_0. To explicitly calculate this value we need to now the distribution of the data in case the alternative hypothesis is true.

We exemplify these concepts with the help of the binomial test. Given the hypothesis that a feature F is binomially distributed with a success probability of $\pi = 0.999$, Figure A.2 shows the probability distribution for $n = 100,000$ samples. If we now observe a count for a feature we can easily check, how probable this would be given our hypothesis, that $F \sim b_{n,\pi}$. For big n the binomial distribution can be approximated by the normal distribution with mean $n\pi$ and variance $n\pi(1-\pi)$ plotted as red line in Figure A.2. We can see that observing a count of 99860 or smaller is highly improbable.

The tests used in the CVST algorithm are common tools in the field of statistical data analysis and in contrast to the binomial test are non-parametric test, since the algorithm makes no distributional assumptions whatsoever. Here we give a short summary based on the Heckert and Filliben [2003] and cast the notation into the CVST framework context. Both methods deal with the performance matrix of K configurations with performance values on r data points:

	Data Points			
Configuration	1	2	...	r
1	x_{11}	x_{12}	...	x_{1r}
2	x_{21}	x_{22}	...	x_{2r}
3	x_{31}	x_{32}	...	x_{3r}
$\vdots$	$\vdots$	$\vdots$		$\vdots$
K	x_{K1}	x_{K2}	...	x_{Kr}

Both tests treat similar questions ("Do the K configurations have identical effects?") but are designed for different kinds of data: Cochran's Q test is tuned for binary x_{ij} while the Friedman test acts on continuous values. In the context of the CVST algorithm the tests are used for two different tasks:

1. Determine whether a set of configurations are the top performing ones (step ❶ in the overview Figure 3.5 and the function topConfigurations in Algorithm 2).

2. Check whether the remaining configurations behaved similar in the past (step ❸ in the overview Figure 3.5 and the function similarPerformance in Algorithm 2).

In both cases, the configurations are compared either by the performance on the samples (Point 1 above) or by the behavior on the last w_{stop} traces (Point 2 above) of the remaining configurations. Depending on the learning problem either the Friedman Test for regression tasks or the Cochran's Q test for classification tasks is used in Point 1. In both cases the hypotheses for the tests are as follows:

- H_0: All configurations are equally effective (no effect)

- H_1: There is a difference in the effectiveness among the configurations, i.e., there is at least one configuration showing a significantly different effect on the data points.

A.4.1 Cochran's Q Test

The test statistic T is calculated as follows:

$$T = K(K-1)\frac{\sum_{i=1}^{K} R_i - \frac{M}{K}}{\sum_{i=1}^{r} C_i(K - C_i)}$$

with R_i denoting the row total for the i^{th} configuration, C_i the column total for the i^{th} data point, and M the grand total. We reject H_0, if $T > \chi^2(1-\alpha, K-1)$ with $\chi^2(1-\alpha, K-1)$ denoting the $(1-\alpha)$-quantile of the χ^2 distribution with $K-1$ degrees of freedom and α is the significance level. As Cochran [1950] points out, the χ^2 approximation breaks down for small tables. Tate and Brown [1970] state that as long as the table contains at least 24 entries, the χ^2 approximation will suffice, otherwise the exact distribution should be used which can either be calculated explicitly [see Patil, 1975] or determined via permutation.

A.4.2 Friedman Test

Let $R(x_{ij})$ be the rank assigned to x_{ij} within data point i (i.e., rank of a configuration on data point i). Average ranks are used in the case of ties. The ranks for a configuration at position k are summed up over the data points to obtain

$$R_k = \sum_{i=1}^{r} R(x_{ki}).$$

The test statistic T is then calculated as follows:

$$T = \frac{12}{rK(K+1)} \sum_{i=1}^{K} (R_i - r(K+1)/2)^2.$$

If there are ties, then

$$T = \frac{(K-1)\sum_{i=1}^{K}(R_i - r(K+1)/2)^2}{[\sum_{i=1}^{K}\sum_{j=1}^{r} R(x_{ij})^2] - [rK(K+1)^2]/4}.$$

We reject H_0 if $T > \chi^2(\alpha, K-1)$ with $\chi^2(\alpha, K-1)$ denoting the α-quantile of the χ^2 distribution with $K-1$ degrees of freedom and α being the significance level.

Appendix B

Proofs and Further Analysis

B.1 The Robot Protocol: A PRISMA Example

In order to illustrate the message-building algorithm used by the PRISMA method, we have developed an experiment that eases to understand how the elements learned are integrated with the Markov model. The Robot example is a simple setup where a robot is randomly placed in a room filled with contaminated objects. Its goal is to find these objects and carry them to the base location where they will be eliminated. Unfortunately, it has no view of the position of the objects or itself within the room. Thus, in order to move around and notice if it has found

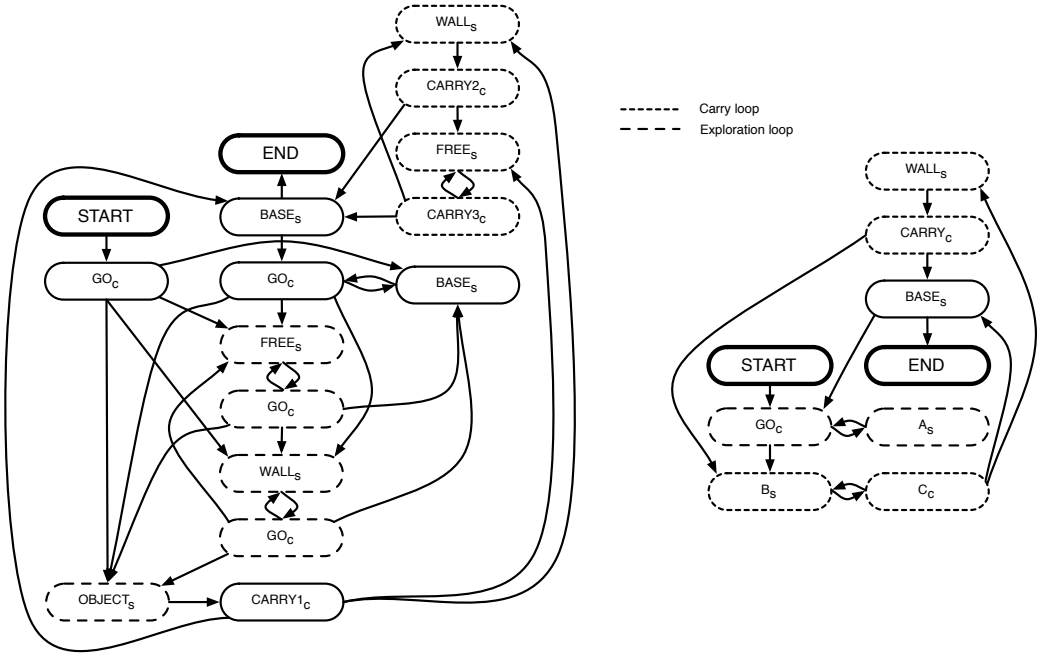


Fig. B.1: The Markov model of the robot protocol and the minimized version of the state model.

an object or reached the limits of the room, it exchanges messages with a room controller.

In our experiment, the robot has a fixed algorithm: At first, it randomly walks through the rectangular room until it finds an object. Then, it goes straight up to the top of the room carrying the object and to the left to reach the base, which is situated at the upper left corner. The object is then destroyed and the robot is dropped again randomly inside the room. The process continues until the robot has found and removed all the objects.

The complete setup has a client-server architecture where the robot communicates with the controller by a simple protocol over the network. These network traces are then analyzed by PRISMA, yielding a robot-honeyclient that is capable of mimicking the behavior of the original robot.

The robot client sends its instructions to the controller using the messages `GO <dir>` and `CARRY <object> <dir>`, where possible directions are `UP`, `RIGHT`, `DOWN` and `LEFT`. The controller server responds with the following status messages after each action of the robot: `WALL`, `FREE`, `BASE` and `OBJECT <object>`, where `<object>` denotes the id of the object.

B.1.1 Markov Model

Figure B.1 presents the Markov model of the robot protocol, which has been extracted from the traces. As intended for this example and for completeness, every possible message is present in the pool of simulated communication data. Therefore, the model represents the complete robot protocol. The lower part of the Markov chain models the communication between client and server during the exploration phase of the robot, while the upper part models the loop where the robot is carrying the object to the base. The subindex indicates the active side of the communication, client or server, in each state. The right side of Figure B.1 depicts the simplified model obtained after the minimization algorithm has been applied to the original model. States labeled A, B and C are meta-states resulting from the abstraction of several states in the original model and are involved in each of the described phases. Meta-state A represents the behavior of the robot client during the exploration phase while meta-states B and C are part of the carrying loop.

B.1.2 Templates and Rules

The different templates associated with each state of the model are inferred from the traces obtained by simulation: The number of runs is specified as an input parameter. Each run requires as many sessions to complete as objects are placed in the room and each session is formed by an arbitrary number of messages as a result of the random direction of the movement.

Table B.1 contains examples of templates and rules which show how the search algorithm is implemented by the model: When an object is found by the robot, the room controller builds a message with the format shown in template 13 in state $\mathrm{OBJECT_S}$. Following the only possible transition in the model, the next state is $\mathrm{CARRY1_C}$, where the client constructs the message using the format of template

State	ID	Format
OBJECT_S	13	`OBJECT` □
CARRY1_C	11	`CARRY` □ `UP`
CARRY2_C	10	`CARRY` □ `LEFT`
CARRY3_C	2	`CARRY` □ □
FREE_S	5	`FREE`

(a) Templates

Trans.	Type	Src. ID	Src. Field	Dest. Field
`3;13;11`	*Copy*	13	0	0
`11;0;10`	*Copy*	11	0	0
`2;5;2`	*Copy*	2	0	0
`2;5;2`	*Copy*	2	1	1

(b) Rules

Tab. B.1: Selected templates and rules from the robot example.

11 and the rule `3;13;11`. This rule indicates that the data in the field 0 must be copied to the field 0 of the current template. This results in the message: `CARRY <object> UP`. When the robot has hit the upper wall, it builds a message in state CARRY2_C according to template 10 and rule `11;0;10`. Now the robot is carrying the object to the `LEFT` in state CARRY3_C until it finds the base. The object and the direction that must be followed are introduced in the message format of template 2 associated to this state by using the rules with transitions `2;5;2`.

B.2 Non-Negative Matrix Factorization via Alternating Least Squares

In this section we give a concrete presentation of our non-negative matrix factorization (NMF) algorithm which exploits replicated values in the data matrix to decrease the runtime or allows the calculation of even very big data sets in memory. NMF models the data $A \in \mathbb{R}^{\tilde{f},N}$ with two matrices $B \in \mathbb{R}^{\tilde{f},e}, C \in \mathbb{R}^{e,N}$ as shown in Equation (2.2), repeated here for convenience:

$$A \approx BC \quad \text{with } (B,C) \quad = \quad \arg\min_{B,C} \|A - BC\| \\ \text{s.t. } b_{ij} \geqslant 0,\, c_{jn} \geqslant 0\,.$$

We have derived the inner workings of the replicate-aware version of NMF in Section 2.1.3 and describe the resulting algorithm in Section B.2.1 for the sake of completeness. We give the heuristic, how to automatically choose the inner dimension by a data-driven procedure in Section B.2.2, and discuss a useful initialization scheme for NMF in Section B.2.3.

B.2.1 Non-Negative Matrix Factorization with Replicates

The function $\text{RRbyCV}(Y, X, W)$ estimates the optimal λ parameter for the ridge regression problem $\min_\beta \|Y - X\beta\|^2 + \lambda\|\beta\|^2$ by 5-fold cross-validation.

Algorithm 3 Replicate-Aware NMF

function nmfWithReplicates(A, e, B, W)
  N = number of columns in A
  $\tilde{f}$ = number of rows in A
  if $B = \mathbf{0}$ **then**
    $b_{ij} = |\mathcal{N}(0,1)|$ ▷ Fill B with normally distr. data
  if $W = \mathbf{0}$ **then**
    $W = I_{N,N}$
  $err = \infty$
  while $|err - \frac{1}{2}\|A - BC\|^2| < \epsilon$ **do**
    $err = \frac{1}{2}\|A - BC\|^2$
    $\lambda = \mathrm{RRbyCV}(A, B, \mathbf{0})$
    $C = (B^\top B + \lambda I_{e,e})^{-1}(B^\top A)$
    $C[C < 0] = 0$ ▷ Set all negative coordinates to 0
    $\lambda = \mathrm{RRbyCV}(A^\top, C^\top, W)$
    $B = (AWC^\top)(CWC^\top + \lambda I_{e,e})^{-1}$
    $B[B < 0] = 0$ ▷ Set all negative coordinates to 0
    $B = B\,\mathrm{diag}(1/\|b_1\|, 1/\|b_2\|, \ldots, 1/\|b_e\|)$ ▷ Normalize columns of B
    $C = C\,\mathrm{diag}(\|b_1\|, \|b_2\|, \ldots, \|b_e\|)$

B.2.2 Estimating the Inner Dimension

The inner dimension e can be chosen according to an argument in Schmidt [1986]: The ordered eigenvalues of the data matrix can be split into a part which is actually contributing to the real signal and a noise part. If we estimate the eigenvalues λ_i on the original data matrix A and the eigenvalues $\widehat{\lambda}_i$ on a scrambled version $\widehat{A}$, where we randomize the features for each message and add confidence intervals to the eigenvalues $\lambda_i, \widehat{\lambda}_i$ according to Jolliffe [1986], we can pick the last index as inner dimension e, in which the confidence intervals $\lambda_e, \widehat{\lambda}_e$ do not overlap.

B.2.3 Initializing the Non-Negative Matrix Factorization

We have seen in Section B.2.1 that the replicate-aware non-negative matrix factorization can be initialized with a suitable starting value for B. In a similar vein to Boutsidis and Gallopoulos [2008] we propose to initialize the NMF algorithm with positive and negative parts of a preceding SVD as shown in Algorithm 4.

Algorithm 4 Initial base for replicate-aware NMF

function generateInitialBase(A, e)
  $A = HDV^\top$ ▷ SVD of A
  $B = H[, 1 : \lfloor \frac{e}{2} \rfloor]$ ▷ Choose the first $\lfloor \frac{e}{2} \rfloor$ components
  $\bar{B} = -B$
  $\bar{B}[\bar{B} < 0] = 0$ ▷ Set negative values to 0
  return $[\bar{B}, B + \bar{B}]$ ▷ Return parts combined as one matrix

B.3 Proof of Convergence Theorem of CVST

Recall (see Equation 3.1) that $e_n(c)$ is the expected error of parameter configuration c. We first prove that we have uniform convergence over finite sets of candidate configurations if the error for individual configurations converges. Let $\varepsilon > 0$, then

$$\begin{aligned} P\left\{\max_{c\in C} |e_n(c) - e(c)| > \varepsilon\right\} &= P\left(\bigcup_{c\in C} |e_n(c) - e(c)| > \varepsilon\right) \\ &\leqslant \sum_{c\in C} P\{|e_n(c) - e(c) > \varepsilon|\} \to 0 \end{aligned}$$

since for each fixed c, $e_n(c) \to e(c)$ in probability.

For the proof of the second statement (convergence of the minimum), let $\varepsilon = \max_{c\in C} |e_n(c) - e(c)|$, then

$$\begin{aligned} e(c_n^*) - e(c^*) &= e(c_n^*) - e_n(c_n^*) + e_n(c_n^*) - e(c^*) \\ &\leqslant \varepsilon + e_n(c_n^*) - e(c^*) \\ &\leqslant \varepsilon + e_n(c^*) - e(c^*) \\ &\leqslant \varepsilon + \varepsilon = 2\varepsilon, \end{aligned}$$

where the first and third inequality hold because of the uniform bound on the error, and the second one because c_n^* is the minimizer of e_n.

Convergence in probability follows because

$$P\{e(c_n^*) - e(c^*) > \varepsilon\} \leqslant P\{\max_{c\in C} |e_n(c) - e(c)| > \frac{\varepsilon}{2}\} \to 0.$$

Finally, for the third statement (convergence for subsets), we start by using the same argument as for the second statement,

$$\begin{aligned} e_n(c_m^*) - e_n(c_n^*) &\leqslant 2 \max_{c\in C} |e_m(c) - e_n(c)| \\ &\leqslant 2 \max_{c\in C} |e_m(c) - e(c)| + 2 \max_{c\in C} |e(c) - e_n(c)| \\ &=: 2m_m + 2m_n. \end{aligned}$$

Now fix some $\delta, \varepsilon > 0$. First of all, note that

$$2m_m + 2m_n \geqslant \varepsilon \quad \text{implies} \quad 2m_m \geqslant \frac{\varepsilon}{2} \vee 2m_n \geqslant \frac{\varepsilon}{2}.$$

Then,

$$P\{2m_m + 2m_n \geqslant \varepsilon\} \leqslant P\{m_m \geqslant \varepsilon/4\} + P\{m_n \geqslant \varepsilon/4\}.$$

As already shown, these probabilities converge to zero, such that there exists an l such that for all $n, m \geqslant l$,

$$P\{m_m \geqslant \varepsilon/4\} \leqslant \frac{\delta}{2}, \text{ and } P\{m_n \geqslant \varepsilon/4\} \leqslant \frac{\delta}{2}.$$

Therefore,

$$P\{e_n(c_m^*) - e_n(c_n^*) \geqslant \varepsilon\} \leqslant P\{m_m \geqslant \varepsilon/4\} + P\{m_n \geqslant \varepsilon/4\} \leqslant \delta,$$

for all $m, n \geqslant l$, in particular for $l \leqslant m \leqslant n$.

B.4 Proof of Safety Zone Bound of CVST

In this section we prove the safety zone bound of Section 3.3.1. We will follow the notation and treatment of the sequential analysis as found in the original publication of Wald [1947], Sections 5.3 to 5.5. First of all, Wald proves in Equation 5:27 that the following approximation holds:

$$\mathrm{ASN}(\pi_0, \pi_1 | \pi = 1.0) = \frac{\log \frac{1-\beta_l}{\alpha_l}}{\log \frac{\pi_1}{\pi_0}}.$$

The minimal $\mathrm{ASN}(\pi_0, \pi_1 | \pi = 1.0)$ is therefore attained if $\log \frac{\pi_1}{\pi_0}$ is maximal, which is clearly the case for $\pi_1 = 1.0$ and $\pi_0 = 0.5$, which holds by construction. So we get the lower bound of S for a given significance level α_l, β_l:

$$S \geqslant \left\lceil \log \frac{1-\beta_l}{\alpha_l} / \log 2 \right\rceil.$$

The lower line L_0 of the graphical sequential analysis test as exemplified in Figure 3.5 is defined as follows (see Equation 5:13 - 5:15):

$$L_0 = \frac{\log \frac{\beta_l}{1-\alpha_l}}{\log \frac{\pi_1}{\pi_0} - \log \frac{1-\pi_1}{1-\pi_0}} + n \frac{\log \frac{1-\pi_0}{1-\pi_1}}{\log \frac{\pi_1}{\pi_0} - \log \frac{1-\pi_1}{1-\pi_0}}.$$

Setting $L_0 = 0$, we can get the intersection of the lower test line with the x-axis and therefore the earliest step s_{safe}, in which the procedure will drop a constant loser configuration. This yields

$$\begin{aligned} s_{\text{safe}} &= -\frac{\log \frac{\beta_l}{1-\alpha_l}}{\log \frac{\pi_1}{\pi_0} - \log \frac{1-\pi_1}{1-\pi_0}} \Big/ \frac{\log \frac{1-\pi_0}{1-\pi_1}}{\log \frac{\pi_1}{\pi_0} - \log \frac{1-\pi_1}{1-\pi_0}} \\ &= -\frac{\log \frac{\beta_l}{1-\alpha_l}}{\log \frac{1-\pi_0}{1-\pi_1}} = \frac{\log \frac{\beta_l}{1-\alpha_l}}{\log \frac{1-\pi_1}{1-\pi_0}} = \frac{\log \frac{\beta_l}{1-\alpha_l}}{\log 2 - \sqrt[S]{\frac{1-\beta_l}{\alpha_l}}}. \end{aligned}$$

The last equality can be derived by inserting the closed form of π_1 given $\pi_0 = 0.5$:

$$S = \mathrm{ASN}(\pi_0, \pi_1 | \pi = 1.0) = \frac{\log \frac{1-\beta_l}{\alpha_l}}{\log \frac{\pi_1}{\pi_0}} = \frac{\log \frac{1-\beta_l}{\alpha_l}}{\log 2\pi_1} \quad \Leftrightarrow \quad \pi_1 = \frac{1}{2} \sqrt[S]{\frac{1-\beta_l}{\alpha_l}}.$$

Setting s_{safe} in relation to the maximal number of steps S yields the safety zone bound of Section 3.3.1.

B.5 False Negative Rate of CVST for Underestimated Change Point

In Section 3.3.1 we have investigated how the CVST algorithm performs if the experimenter was able to ensure a stable regime via the safety zone. Now, we go even a step further and look at the performance if the experimenter underestimated the

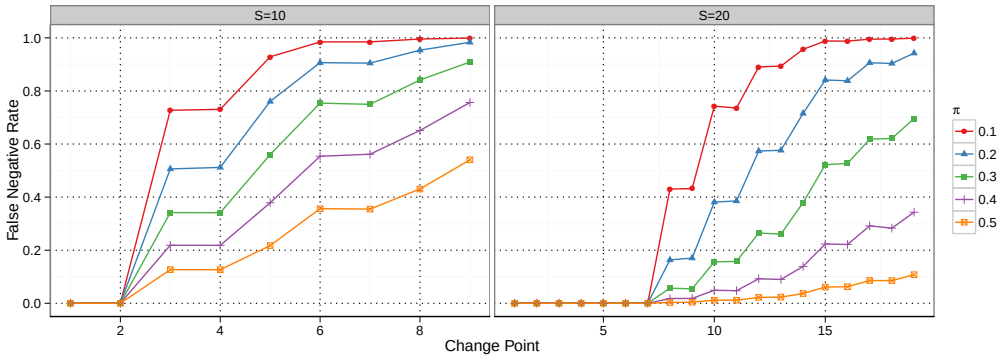


Fig. B.2: False negatives generated with the open sequential test for non-stationary configurations, i.e., at the given change point the Bernoulli variable changes its π_{before} from the indicated value to 1.0.

change point s_{cp}. To get insight into the dropping rate we simulate those switching configurations by independent Bernoulli variables which change their success probability π from a chosen $\pi_{\text{before}} \in \{0.1, 0.2, \ldots, 0.5\}$ to a constant 1.0 at a given change point. This behavior imitates the behavior of a switching configuration which starts out as a loser (i.e. up to the change point the trace will consist more or less of zeros) and after enough data is available turns into a constant winner.

The relative loss of these configurations for 10 and 20 steps is plotted in Figure B.2 for different change points. The figure reveals our theoretical findings of Lemma 2 showing the corresponding *safety zone* for the specific parameter settings: For instance for $\alpha_l = 0.01$ and $\beta_l = 0.1$ and $S = 10$ steps, the safety zone amounts to 0.27×10, meaning that if the change point for all switching configurations occurs at step one or two, the CVST algorithm would not suffer from false positives. Similarly, for $S = 20$ the safety zone is $0.39 \times 20 = 7.8$. These theoretical results are confirmed in our simulation study, where the false negative rate is zero for sufficiently small change points for the open variant of the test. After that, there are increasing probabilities that the configuration will be removed. Depending on the success probability of the configuration before the change point, the resulting false negative rate ranges from mild for $\pi = 0.5$ to relatively severe for $\pi = 0.1$. The later the change point occurs, the higher the resulting false negative rate will be. Interestingly, if we increase the total number of steps from 10 to 20, the absolute values of the false negative rates are significantly lower. So even when the experimenter underestimates the actual change point, the CVST algorithm has some extra room which can even be extended by increasing the total number of steps.

B.6 Proof of Computational Budget of CVST

For the size N of the whole data set and a cubic learner, resulting in a learning time of $t = N^3$, one observes that learning on a proportion of size $\frac{i}{S}N$ takes about $\frac{i^3}{S^3}t$ time. Via construction one has to learn on all k parameter configurations in each step before hitting $s_r \times S$ and on $K \times (1-r)$ parameter configurations with drop ratio r afterwards. Thus, the entirely needed computation time is given by

$$K \times (1-r) \sum_{i=1}^{s_r \times S} \frac{i^3}{S^3} t + K \times r \sum_{i=1}^{S} \frac{i^3}{S^3} t$$

which should be smaller than the given time budget T.

Making use of the equality $\sum_{i=1}^{j} i^3 = \frac{j^2(j+1)^2}{4}$ one can reformulate the inequality:

$$\begin{aligned} T &\geqslant \frac{t \times K(1-r)}{S^3} \frac{s_r \times S^2(s_r \times S + 1)^2}{4} + \frac{t \times K \times r}{S^3} \frac{S^2(S+1)^2}{4} \\ &= \frac{t \times K}{S} \Big[(1-r) \frac{s^2(s_r \times S + 1)^2}{4} + r \frac{(S+1)^2}{4} \Big] \end{aligned}$$

It is obvious that this inequality is quadratic in the variable S which can be solved by bringing the above inequality in standard form:

$$\begin{aligned} & 0 \geqslant \Big[(1-r) \frac{s_r^2(s_r \times S + 1)^2}{4} + r \frac{(S+1)^2}{4} \Big] - \frac{T \times S}{t \times k} \\ \Leftrightarrow \; & 0 \geqslant \frac{(1-r)s_r^4 + r}{4} S + 1 s_r^2 + \Big[\frac{(1-r)s_r^3 + r}{2} - \frac{T}{t \times k} \Big] S + \frac{(1-r)s_r^2 + r}{4} \\ \Leftrightarrow \; & 0 \geqslant S s_r^2 + 2 \frac{t \times k(1-r)s_r^3 + t \times k \times r - 2T}{((1-r)s_r^4 + r)t \times k} S + \frac{(1-r)s_r^2 + r}{(1-r)s_r^4 + r}. \end{aligned}$$

Substituting $a = \frac{t \times k(1-r)s_r^3 + t \times k \times r - 2T}{((1-r)s_r^4 + r)t \times k}$ and $b = \frac{(1-r)s_r^2 + r}{(1-r)s_r^4 + r}$ above is equivalent to:

$$S = -a + y, \; y \in \left\{ -\sqrt{a^2 - b}, +\sqrt{a^2 - b} \right\}.$$

For the sake of a meaningful step amount, i.e. $S > 0$ and furthermore S as large as possible we choose it as

$$S = \left\lfloor -a + \sqrt{a^2 - b} \right\rfloor.$$

Note that S is a function of the parameter s. Since obviously $b \geqslant 0$ holds a must be negative in order to gain a positive step amount. Furthermore the root has to be solvable. So the following constraints on s_r have to be made:

$$\begin{aligned} &(1) \qquad 2T \geqslant t \times k(1-r)s_r^3 + t \times k \times r \\ &(2) \qquad a^2 \geqslant b. \end{aligned}$$

B.7 Example Run of CVST Algorithm

In this section we give an example of the whole CVST algorithm on one *noisy sinc* data set of $n = 1,000$ data points with intrinsic dimensionality of $d = 2$. The CVST algorithm is executed with $S = 10$ and $w_{\text{stop}} = 4$. We use a ν-SVM [Schölkopf et al., 2000] and test a parameter grid of $\log_{10}(\sigma) \in \{-3, -2.9, \ldots, 3\}$ and $\nu \in \{0.05, 0.1, \ldots, 0.5\}$. The procedure runs for 4 steps after which the early stopping rule takes effect. This yields the following trace matrix (only remaining configurations are shown):

	n = 90	**n = 180**	**n = 270**	**n = 360**
$\log_{10}(\sigma) = -2.3, \nu = 0.35$	0	0	1	0
$\log_{10}(\sigma) = -2.3, \nu = 0.40$	0	1	1	0
$\log_{10}(\sigma) = -2.3, \nu = 0.45$	0	1	0	1
$\log_{10}(\sigma) = -2.2, \nu = 0.30$	0	1	0	0
$\log_{10}(\sigma) = -2.2, \nu = 0.35$	0	1	1	0
$\log_{10}(\sigma) = -2.2, \nu = 0.40$	0	1	1	1
$\log_{10}(\sigma) = -2.2, \nu = 0.45$	0	1	1	1
$\log_{10}(\sigma) = -2.2, \nu = 0.50$	0	0	1	1
$\log_{10}(\sigma) = -2.1, \nu = 0.35$	0	1	1	1
$\log_{10}(\sigma) = -2.1, \nu = 0.40$	0	1	1	1
$\log_{10}(\sigma) = -2.1, \nu = 0.45$	0	1	1	1
$\log_{10}(\sigma) = -2.1, \nu = 0.50$	1	0	1	1
$\log_{10}(\sigma) = -2.0, \nu = 0.50$	0	0	1	1

The corresponding mean square errors of the remaining configurations after each step are shown in the next matrix. Based on these values, the winning configuration, namely $\log_{10}(\sigma) = -2.1, \nu = 0.40$ is chosen:

	n = 90	**n = 180**	**n = 270**	**n = 360**
$\log_{10}(\sigma) = -2.3, \nu = 0.35$	0.0370	0.0199	0.0145	0.0150
$\log_{10}(\sigma) = -2.3, \nu = 0.40$	0.0362	0.0197	0.0146	0.0146
$\log_{10}(\sigma) = -2.3, \nu = 0.45$	0.0356	0.0197	0.0146	0.0144
$\log_{10}(\sigma) = -2.2, \nu = 0.30$	0.0365	0.0195	0.0146	0.0148
$\log_{10}(\sigma) = -2.2, \nu = 0.35$	0.0351	0.0193	0.0142	0.0145
$\log_{10}(\sigma) = -2.2, \nu = 0.40$	0.0345	0.0194	0.0143	0.0141
$\log_{10}(\sigma) = -2.2, \nu = 0.45$	0.0340	0.0193	0.0143	0.0140
$\log_{10}(\sigma) = -2.2, \nu = 0.50$	0.0332	0.0200	0.0145	0.0138
$\log_{10}(\sigma) = -2.1, \nu = 0.35$	0.0353	0.0194	0.0144	0.0142
$\log_{10}(\sigma) = -2.1, \nu = 0.40$	0.0343	0.0195	0.0142	0.0138
$\log_{10}(\sigma) = -2.1, \nu = 0.45$	0.0340	0.0197	0.0140	0.0138
$\log_{10}(\sigma) = -2.1, \nu = 0.50$	0.0329	0.0199	0.0142	0.0137
$\log_{10}(\sigma) = -2.0, \nu = 0.50$	0.0351	0.0204	0.0145	0.0137