Features and Machine Learning Systems for Structured and Sequential Data

vorgelegt von
Dipl.Inf.
Guido Schwenk

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

Doktor der Ingenieurwissenschaften
- Dr.-Ing. -

genehmigte Dissertation

Tag der wissenschaftlichen Aussprache: 18. März 2019

Berlin 2019

To my beloved wife and our children, for their never ending support.

Acknowledgements

I would like to thank Prof. Dr. Klaus-Robert Müller and Prof. Dr. Konrad Rieck for their enduring optimism and encouragement.

Summary

Modern web and communication technology relies heavily on sequential and structured data for its process execution and communication protocols. Due to its complex properties, a manual analysis and detection of problems on this data is too time-consuming and expensive, and hence not feasible. As a consequence, features and automatic learning systems on this type of data are highly sought after.

To address these issues, the thesis proposes features and systems for learning on structured, sequential and temporal data, both in abstract and in concrete form, with a focus on analyses in the fields of IT security and Quality of Service, on the data domains of analysis data of malware binaries and JavaScript code, as well as on mobile network communication data. The proposed features and feature combinations cover various statistical, non-behavioral and behavioral, stateless, stateful, structural and temporal concepts, and are used individually and in a complementary manner, e.g. via hierarchical or ensemble approaches. The proposed learning systems are evaluated against competitive approaches, where they outperform commonly used and state-of-the-art methods, including approaches using neural networks.

Specific practically relevant aspects are also addressed in depth, like high levels of automation to extend the scope of the system application, different re-training procedures, or the calibration of metrics relevant for the specific domain. To improve the interpretability of the system processes and their results to increase the system reliability and its level of trust, different visualization approaches are proposed, focussing on interpretable and transparent feature projections and relevance analyses. These additional discussions on the proposed ideas further support a potential adaptation of the proposed ideas to concrete application scenarios.

Zusammenfassung

Moderne Internet und Kommunikationstechnologien nutzen sequentielle und strukturierte Daten zur Ausführung ihrer Prozesse und Kommunikationsprotokolle. Aufgrund deren komplexer Eigenschaften ist eine manuelle Erkennung und Analyse von Problemen auf diesen Daten zu zeitaufwendig und teuer und daher oft nicht realisierbar. Infolgedessen sind automatisierte Lernsysteme, die auf solchen Daten arbeiten und dies ermöglichen, sehr gefragt.

Die vorliegende Dissertation adressiert dies in mehrfacher Hinsicht. So werden Eigenschaften, Systeme und Merkmale von strukturierten, sequentiellen und temporalen Daten diskutiert, sowohl in abstrakter wie auch konkreter Form am Beispiel von Analysendaten von JavaScript Code und Schadsoftware im Bereich der IT Sicherheit, sowie auf Mobilfunkkommunikationsdaten zu Zwecken der Qualitätssicherung. Es werden verschiedene individuelle wie auch kombinierte statistische, verhaltensbasierte, zustandslose, zustandsbasierte, strukturelle sowie temporale Merkmalsarten eingeführt und analysiert. Dabei werden deren Eigenschaften sowohl im individuellen Gebrauch wie auch im Verbundgebrauch analysiert, beispielsweise in Form von hierarchischen Merkmalsverbänden oder Ensemble-Ansätzen. Die Klassifikationsleistungen und Merkmalseigenschaften der vorgestellten Lernsysteme werden im Rahmen umfangreicher Evaluationen mit konkurrierenden Ansätzen verglichen. Dabei zeigen sich sehr gute Ergebnisse der vorgestellten Methoden, selbst im Vergleich zu state-of-the-art Methoden wie neuronalen Netzwerken.

Zusätzlich werden praktisch relevante Aspekte der besprochenen Probleme adressiert, um deren Potential einer realen Anwendung zu erhöhen. Dazu gehören beispielsweise ein hoher Grad an Automatisierung der vorgeschlagenen Systeme, verschiedene Trainingsprozeduren, sowie Möglichkeiten der Kalibrierung von Metriken, die für das besprochene Anwendungsgebiet relevant sind. Auch die Möglichkeiten der Interpretierbarkeit und Transparenz der vorgeschlagenen Systeme werden besprochen und mit verschiedenen Methoden adressiert, um dadurch das Vertrauen in die vorgestellten automatisierten Lernsysteme zu erhöhen. Unter anderem werden dazu verschiedene Visualisierungsmethoden für Daten und Merkmale, sowie Möglichkeiten der Ergebnisrelevanzanalyse vorgestellt.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Motivation

Modern internet and communication technologies rely heavily on the processing of functionally structured textual and numerical data, like executed code or data transmitted in accordance with network communication protocols. Necessarily, the structures contained in this data are semantically relevant for the process execution. As these processes also require a sequential execution of its functional components, this data can also be viewed as sequential data. As a result of the complexity of these processes, practical applications need to address various related problems, like the detection of malicious code or communication to enable a sufficient level of IT security, or the classification and validation of network communication to enable a higher quality of the respective communication service. While manual problem analysis on such data is possible, it is time-consuming and expensive. Consequently automatically learning systems addressing such problems are highly sought after. To enable such systems, data representations as well as learning methods are required which allow an efficient processing of this type of structured, sequential data. Additionally, these data representations and learning methods also need to be combined in a highly automated manner, reducing the requirement for manual processing steps. And finally these systems should also be transparent, enabling the interpretation of the processing steps and the results. These objectives build the *practical motivation* of this thesis, i.e. is to propose and analyze features, learning methods and complete systems to automatically learn, detect, predict and interpret problems based on sequential and structured code and communication data. Consequently, it aims at answering the following questions:

- Which properties are relevant for structural and sequential data of network communication and code?
- Which types of features are best suited to represent those properties - and how are they extracted effectively?
- Which learning methods and approaches are best suited for solving the different relevant objectives?

- What is needed to achieve highly automated systems utilizing these features and learning methods?

As each thesis chapter focusses on slightly different types of data and use-cases, detailed discussions of the related research are provided in each respective chapter. Generally speaking though, learning on structured as well a structured sequential data is relevant in various research areas. As such relevant methods of the research field of Natural language processing [BPX$^+$07, WM12] are applied to Sequence learning [LMP01, Gra12] or to solve IT security objectives like intrusion detection [Rie09] or malware classification [RTWH11]. In the more recent field of process mining [VDAADM$^+$11, ERF16] this is extended to state-based event sequences with additional temporal features. While the research provided in those areas addresses some of the problems we will face within this thesis, the complex nature of the utilized data often requires deviating adaptations of the utilized methods and approaches.

Data Domains

The thesis discusses general properties of sequential structural and temporal data, to allow an abstraction of the proposed ideas to other research areas. But to highlight the practical relevance and consequences of the proposed ideas, it also focuses on a set of concrete data domains. In the first half of the thesis these are the practical aspects of different types of concrete features in the domain of IT security, namely the detection of covert malicious HTTP communication in Chapter 2, and the detection of malicious Javascript code in Chapter 3. In the second half the focus is extended to more abstract analyses of different types of features and feature concepts, with a focus on the detection and classification of erroneous process behavior in internet communication protocols in Chapter 4 and in specifically preprocessed mobile network communication protocol data in Chapter 5. These focus points of the different chapters are illustrated in Figure 1.1.

Properties and Features

To provide a broad view on the data properties and their relevant features, various statistical, behavioral, non-behavioral, stateless, stateful, structural and temporal features are discussed, both for their individual representations, as well as for different ways of combining them via hierarchical features, feature concatenations and ensemble approaches. Thus Chapter 2 starts with introducing different concrete features like length and entropy values, and it also introduces structural and temporal features representing highly specific properties. Chapter 3 builds on this idea of concrete numerical features, and further extends it by introducing token n-grams in the problem domain. These are then further developed into stateless features and compared against stateful features in Chapter 4, before they are finally extended to the temporal structural feature domain in Chapter 5, where they are compared against different sequential and non-sequential types of features.

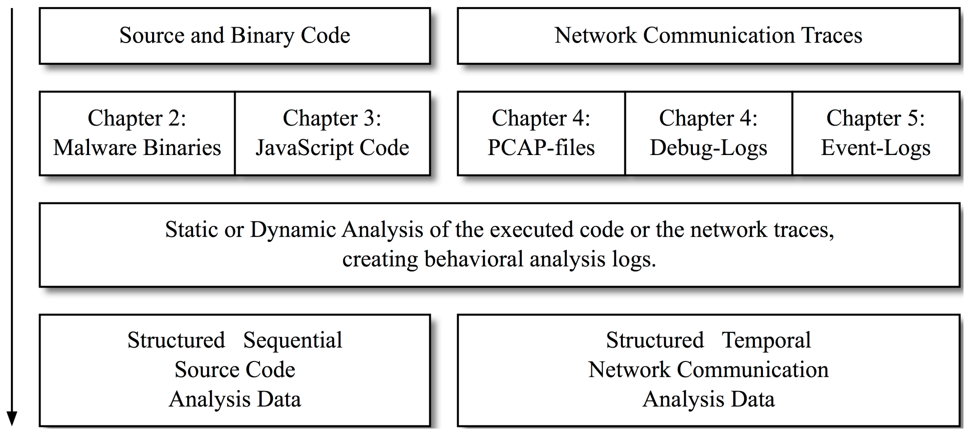

Source and Binary Code
Network Communication Traces
Chapter 2: Malware Binaries
Chapter 3: JavaScript Code
Chapter 4: PCAP-files
Chapter 4: Debug-Logs
Chapter 5: Event-Logs
Static or Dynamic Analysis of the executed code or the network traces, creating behavioral analysis logs.
Structured Sequential Source Code Analysis Data
Structured Temporal Network Communication Analysis Data

Figure 1.1: Overview: From Source Data to Analysis Data

Learning and Detection Systems

To solve the concrete use case objectives and to enable a better analysis the capabilities of the different feature spaces, complete machine learning systems are proposed, allowing for extensive evaluations of the feature performances in their context. The proposed systems and features are evaluated against competitive approaches, where they outperform commonly used and state-of-the-art methods, and even methods based on neural networks. Additionally specific practically relevant aspects are addressed in depth. As such achieving a high level of automation is relevant to reduce the amount of manual system processes, largely extending the scope of the system application. As different application domains require systems which allow for the calibration of system relevant metrics, such calibration procedures are proposed as well. Finally also the interpretation of the system processes and their results need to be enabled, to increase the system reliability and its achieved level of trust. This is achieved through different proposed analysis and visualization approaches, focussing on transparent feature projections and a transparent feature relevance analysis.

Thesis Contributions

As a result of these analyses, this thesis provides the following practical and theoretical contributions:

- Analysis of the *properties* of structured, sequential data, specifically of network communication and code.
- Proposal of novel *features* or feature combinations to represent those properties.
- Proposal of novel ways of combining *learning methods* to achieve the respective classification and detection objectives.

- Proposal and evaluation of highly *automated systems* for solving specific practically relevant learning problems.
- In depth *evaluation* of the proposed features, learning methods and systems, competitively compared against approaches used in related research areas like IT security, sequence learning and process learning.
- Proposal of methods and graphical representations to increase the transparence and *interpretability* of the system and the obtained results.

1.1 Thesis Roadmap

Chapter 2 proposes and evaluates different individual and combined statistical features, which are also used complementarily in a hierarchical manner. A complete machine learning system for the detection of covert and tunneled outbound HTTP communication is proposed, preventing malicious activity and hardening networks against malware proliferation. The proposed unsupervised detection system allows for a calibration of the false positive rate, as required by the respective potential application scenario. It is evaluated on the analysis data of real-world malware binaries, collected over 90 days of user-data, achieving very good detection performances.

Chapter 3 proposes and evaluates different behavioral and non-behavioral features, calculated for dynamically and statically analyzed data for an empirical study of a fully automated system for collecting, analyzing and detecting malicious JavaScript code. The proposed supervised detection system allows for a calibration of the false positive rate, as required by the respective potential application scenario, as well as for a complementary combination of the utilized detectors. It is evaluated on a large dataset of benign and malicious webpages, achieving very good detection performances.

Chapter 4 proposes and evaluates different stateless, stateful and their complementarily combined features. The proposed supervised classification system aims for the classification of mobile validation data for service quality and system dependability of mobile communication data, and is evaluated on two extensive data sets of real-world data, for which two competitive data representations are analyzed for their individual and combined performance and general applicability, achieving very good classification results. The proposed system also allows for a relevance ranking via the analysis of the trained weight vector, for which a visualization is introduced, allowing the interpretation of the classification results.

Chapter 5 proposes and evaluates structural, temporal and complementarily combined features for the detection and prediction of mobile failure data for service quality and system dependability, for a specific type of mobile network communication data. The proposed supervised classification system is evaluated on real-world data sets, where it performed better than methods used in related work. The system also allows for the calibration of a high precision and effective recall. Additionally different hypotheses for the different feature spaces are analyzed, highlighting the practical relevance of the proposed feature space. Additionally a visualization method for the interpretation of the proposed feature space projection is provided.

1.2 Own Contributions

Chapter 2 is based on the publication "Adaptive Detection of Covert Communication in HTTP Requests" by Guido Schwenk and Konrad Rieck, published in the Proceedings of the European Conference on Computer Network Defense 2011, EC2ND11. My contributions to the described work are extensive, as I set up the virtual network and implemented and conducted the execution of the collected binaries and PDF files, collecting the outbound HTTP communication in the process. Afterwards I also conducted all analyses and evaluations on this data.

Chapter 3 is based on the publication "Autonomous Learning for Detection of JavaScript Attacks: Vision or Reality?" by Guido Schwenk, Alexander Bikadorov, Tammo Krueger and Konrad Rieck, published in the Proceedings of the 2012 ACM CCS Workshop on Artificial Intelligence and Security, AISec 2012. My contributions to the described work are extensive and cover the feature extraction, the application of the detectors and all subsequent evaluations.

Chapter 4 is based on the publication "Classification of Structured Validation Data using Stateless and Stateful Features" by Guido Schwenk, Ralf Pabst and Klaus-Robert Müller, Journal of Computer Communications (Elsevier), 2019. With the exception of the actual data collection, which was done externally, I conducted all of the described work, ranging from the data pre-processing over the definition and extraction of features to the implementation and evaluation of the learning methods and feature types, including all discussed analyses and data visualizations.

Chapter 5 is based on the publication "Feature Spaces and a Learning System for Structural-Temporal Data" by Guido Schwenk, Ben Jochinke and Klaus-Robert Müller, submitted to PloS one, December 2018. With the exception of the actual data collection, which was done externally, I conducted all of the described work, ranging from the data pre-processing over the definition and extraction of features to the definition, implementation and evaluation of the proposed detection and prediction system, including the significance analyses and the proposed data visualizations.

1.3 List of Publications

- "Classification of Structured Validation Data using Stateless and Stateful Features" by Guido Schwenk, Ralf Pabst, Klaus-Robert Müller, Journal of Computer Communications (Elsevier), 2019
- "Feature Spaces and a Learning System for Structural-Temporal Data" by Guido Schwenk, Ben Jochinke, Klaus-Robert Müller, submitted to PloS one, December 2018
- "A Close Look on n-Grams in Intrusion Detection" by Christian Wressnegger, Guido Schwenk, Daniel Arp, Konrad Rieck, Proceedings of the 2013 ACM CCS Workshop on Artificial Intelligence and Security, AISec 2013
- "Autonomous Learning for Detection of JavaScript Attacks: Vision or Reality?" by Guido Schwenk, Alexander Bikadorov, Tammo Krueger, Konrad Rieck, Proceedings of the 2012 ACM CCS Workshop on Artificial Intelligence and Security, AISec 2012
- "Detecting Behavioral and Structural Anomalies in MediaCloud Applications" by Guido Schwenk, Sebastian Lapuschkin, arXiv September 2012
- "Adaptive Detection of Covert Communication in HTTP Requests" by Guido Schwenk, Konrad Rieck, Proceedings of the European Conference on Computer Network Defense 2011, EC2ND11
- "Botzilla: Detecting the "Phoning Home" of Malicious Software" by Konrad Rieck, Guido Schwenk, Tobias Limmer, Thorsten Holz and Pavel Laskov, ACM Symposium on Applied Computing 2010, SAC10

Chapter 2

Adaptive Detection of Covert Communication in HTTP Requests

Summary

The infection of computer systems with malicious software is an enduring problem of computer security. Avoiding an infection in the first place is a hard task, as computer systems are often vulnerable to a multitude of attacks. However, to explore and control an infected system, an attacker needs to establish a communication channel with the victim. While such a channel can be easily established to an unprotected end host in the Internet, infiltrating a closed network usually requires passing an application-level gateway - in most cases a web proxy - which constitutes an ideal spot for detecting and blocking unusual outbound communication.

This chapter introduces DUMONT, a system for detecting covert outbound HTTP communication passing through a web proxy. DUMONT learns profiles of normal HTTP requests for each user of the proxy and adapts to individual web surfing characteristics. The profiles are inferred from a diverse set of features, covering the structure and content of outbound data, and allowing for automatically identifying tunnels and covert channels as deviations from normality. While this approach does not generally rule out sophisticated covert communication, it significantly improves on state-of-the-art methods and hardens networks against malware proliferation. This capability is demonstrated in an evaluation with 90 days of web traffic, where DUMONT uncovers the communication of malware, tunnels and backdoors with few false alarms.

2.1 Introduction

Computer networks face a wide variety of threats from malicious software (*malware*). Just a few years ago, malicious software could be categorized into a few basic classes, but nowadays we are confronted with a plethora of malicious tools developed by an underground economy for monetary gains [e.g., FPPS07, HEF09, SGHSV11]. This malware is characterized by versatile functionality and the capability to take numerous routes to a victim, ranging from malicious documents and shortened links to drive-by downloads and targeted attacks. In practice, detecting and eliminating all these infection vectors has proven to be an intractable task and thus millions of hosts in the Internet are plagued by malicious software.

Once compromised, infected machines are regularly misused for illegal activities, such as gathering personal data, distributing spam messages or conducting attacks against other hosts. All these activities inherently require establishing a communication channel that enables the attacker to retrieve data and control the infected system. Such a channel can be trivially established to an unprotected host, for example, by directly sending network packets as performed by the trojans Storm and Nugache [SDHD07, HSD$^+$08]. As a result, a large body of research has studied methods for detecting direct communication with infected hosts [e.g. GZL08, GPZL08, WBH$^+$09, RSL$^+$10]. However, enterprise and government networks are often shielded from the Internet by an application-level gateway - typically in form of a packet filter and a web proxy - and thus no direct communication with infected machines can be established. In this setting, the malicious software is required to tunnel its communication through the web proxy and there is a need for methods capable of detecting tunneled and covert communication in HTTP.

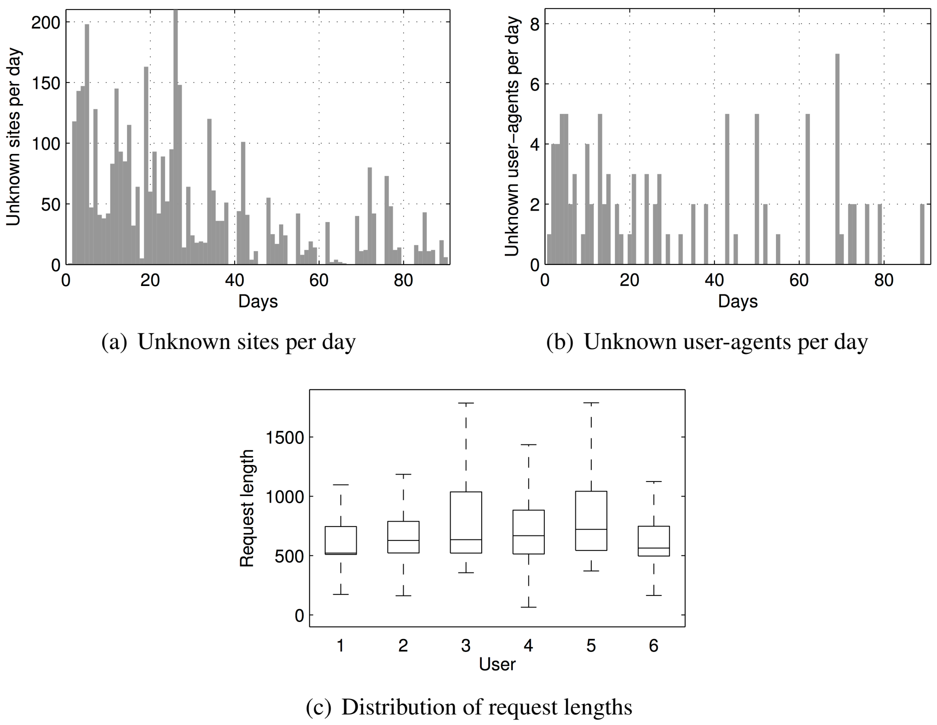
For this purpose this chapter introduces DUMONT, an anomaly detection system for identifying tunneled and covert communication passing through a web proxy. DUMONT learns profiles of normal HTTP requests for each user of the proxy and thereby adapts to the individual web surfing characteristics of each user. The individual profiles are inferred from a diverse set of features, covering the structure and content of outbound data. Using these profiles, tunnels and covert communication of malicious software can be identified as deviations from normality, where respective requests can be put on hold and further investigated before leaving the network. Similarly, DUMONT can be applied for analysis of suspicious files in a sandbox, where it can detect unusual web traffic, for instance, when a spyware program transfers gathered data to a remote host.

Detecting covert channels in the general case is a very ambitious task and clearly DUMONT can not spot arbitrarily sophisticated covert communication, for example, using the timing of requests for encoding information. However, the involved implementation and low transmission rates of such advanced channels render them less attractive for adversaries. In practice DUMONT significantly improves on the detection capabilities of related methods such as WEBTAP [BP04] and raises the bar for malware authors to comprise networks. In an empirical evaluation with 90 days of web traffic from six users, DUMONT allows to identify the majority of malicious software, tunnels and backdoors with a false-positive rate of 0.35%, whereas the rule-based method WEBTAP suffers from over 3% false-positives due to the dynamics of web traffic.

The rest of this chapter is structured as follows: In Section 2.2 the dynamics of HTTP are discussed. The DUMONT system and underlying learning techniques are presented in Section 2.3 and evaluated in Section 2.4. Section 2.5 presents related work and Section 2.6 the conclusion.

2.2 Dynamics of HTTP Communication

The HTTP protocol features a diversity of properties, exploitable to learn something about a user's communication behavior. While most of them look static at a first glance, they show a rather dynamic behavior in practice. When analyzing for example, which web sites a set of users visits during a defined time period, one might consider creating a whitelist of benign web sites sufficient for stopping outbound communication to malicious sites. This assumption is unrealistic, as Figure 2.1(a) illustrates.



(a) Unknown sites per day

(b) Unknown user-agents per day

(c) Distribution of request lengths

Figure 2.1: Examples of the dynamics in the recorded HTTP communication.

Formerly unknown (i.e. first-time seen) benign web sites do always occur, be it through the evolution of the Internet or just through the normal web behavior of the user. Furthermore previously benign web sites might have been infected recently, making them no longer viable for a benign whitelist. For those reasons we do not learn the concrete web site addresses, but model them indirectly using machine learning.

Another dynamic behavior can be observed in the occurrences of HTTP headers. For example, creating a simple whitelist of the appearances of the header *User-Agent* for individual users can provide a means to detect deviations and suspicious combinations (e.g. of the operating system and the web browser). However, this static approach is also not sufficient, as Figure 2.1(b) illustrates. Previously unseen *User-Agents* occur all the time, be it due to changing tools on the client side or simple version changes in the different web clients. A more indirect method of modeling HTTP requests is necessary here as well.

Another question is, whether adaptive learning on the data of individual users provides advantages over learning on an agglomerated dataset of several users together. As Figure 2.1(c) illustrates for the distribution of a single feature of HTTP traffic, namely the lengths of the requests, the same feature may show a different statistical behavior for each user. As learning a representative model of normality requires those features to have consistent statistics, learning on data of individual users is preferable to learning on data of all users combined.

These three examples demonstrate that the dynamics of HTTP communication can hardly be tackled by rule-based methods, such as WEBTAP [BP04]. Hence a learning-based approach to the detection of covert communication in HTTP is applied here, capable of adapting to the individual characteristics of each user.

2.3 The DUMONT System

In the following, the design of our system DUMONT and its inner workings are presented. The selection of features of outbound HTTP traffic is discussed in Section 2.3.1, while the necessary learning method as well as the design of the detector are introduced in Section 2.3.2. The concrete necessities and technical details of the operation are then laid out in Section 2.3.3.

2.3.1 Features of HTTP Requests

According to RFC2616 [FGM$^+$99] an HTTP request starts with a method, e.g. `GET` or `POST`. A method requires an URI, which may include pairs of parameters and values. After the URI, HTTP headers are defined, again consisting of pairs of parameters and values. If a `POST` request is triggered, typically a body of data concludes the request. Additional to features of this data, each request is triggered at a certain point in time, whose features can be stored as well. Based on those elements of HTTP requests 17 descriptive features grouped in 4 semantic sets are extracted. In networks secured with DUMONT, the use of the HTTP method `CONNECT` is to be restricted, as this method implements a standard tunnel protocol. Allowing an unmonitored use of such a communication channel does undermine the objective of DUMONT. Therefore requests using that method are not taken into account.

Length features: The set of length features is depicted in Table 2.1. It describes length values of different parts of the request, such as URI and body, for later detecting deviations from those values.

Feature	Description
l_1	Length of request
l_2	Length of URI
l_3	Total length of URI parameters
l_4	Total length of headers
l_5	Length of request body

Table 2.1: Length features of HTTP requests.

Structural features: The set of structural features is shown in Table 2.2. It contains values describing the structure of an HTTP request by statistical measures, such as the average length of URI parameter names or header values. This perspective allows identifying outbound data otherwise hidden through distribution over different headers or parameters.

Feature	Description
s_1	Average length of URI parameter names
s_2	Average length of URI parameter values
s_3	Average length of header names
s_4	Average length of header values

Table 2.2: Structural features of HTTP requests.

Entropy features: The set of entropy features, depicted in Table 2.3, contains entropy values for different bit widths. These values allow an estimation of the information content in the analyzed request, where the different bit widths cover the request content at different granularity.

Feature	Description
e_1	8-bit entropy of request
e_2	16-bit entropy of request
e_3	24-bit entropy of request
e_4	32-bit entropy of request

Table 2.3: Entropy features of HTTP requests.

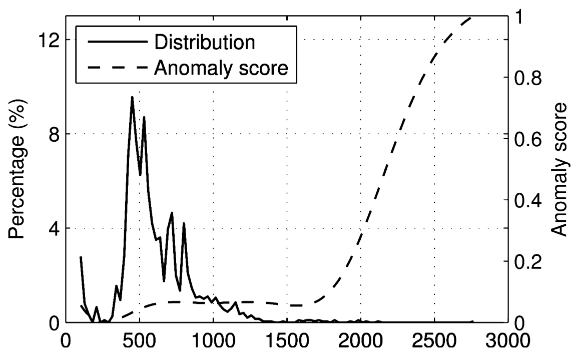
Temporal features: The set of temporal features is illustrated in Table 2.4. These features enable the analysis of temporal traffic characteristics and help to spot unusual communication activity.

Feature	Description
t_1	Number of requests in last minute
t_2	Number of outbound bytes in last minute
t_3	Hour of HTTP request
t_4	Week day of HTTP request

Table 2.4: Temporal features of HTTP requests.

2.3.2 Anomaly Detection

Our system DUMONT makes use of a standard learning technique - the *One-Class SVM* [TD99, SPST$^+$01] - for learning a model of normality for the different HTTP features. Formally, a One-Class SVM describes a hypersphere. This sphere encloses given data with a minimal volume. Anomalies are detected through their distance from the center of the learned sphere, resulting in a high anomaly score. To compensate outliers and noise, as well as to optimize false-positive and detection rates, a soft margin is used. This way not all normal data points are required to reside within the sphere. By using specialized functions, so-called kernels, the sphere can be embedded into a high-dimensional feature space, facilitating the modeling of more complex structures with non-linear representations. In the setup discussed here, Gaussian kernels [MMR$^+$01] are applied to achieve this. More details on support vector machines for one class learning, their optimization problem and decision function, as well as more details on the use of kernels can be found in Chapter 7.1.



(a) Length of requests

Figure 2.2: Frequency distribution and anomaly scores per request length.

A non-linear model for normality is illustrated in Figure 2.2, on the example of the length of requests. The left y-axis shows the frequency of different request lengths for one user and the right y-axis shows a function of the anomaly score computed using a One-Class SVM with Gaussian kernels. In principle longer requests cause a higher anomaly score. However, the local minimum at a request length of 1,600 byte demonstrates the advantage of a non-linear representation to model ranges of normality more subtle than simple upper and lower bounds can do.

Hierarchical detection layers

Another important concept realized in DUMONT is the combination of individual detectors in hierarchical layers. The trained models for individual features respectively (*detection layer 0*) provide the capability to detect covert channels reflected in single features. However, by training models on the combined features of the four feature sets (*detection layer 1*), the detection capability can be further increased to also identify anomalies in the combination of features. This way for example, anomalous requests can be detected, which are normal in both length and entropy, but anomalous in their combination.

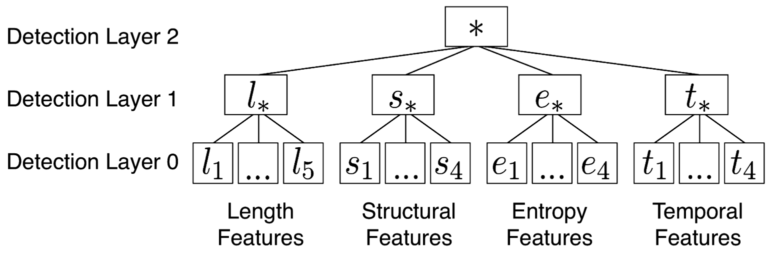


Figure 2.3: Hierarchical detection layers.

The concept is illustrated in Figure 2.3, depicting *detection layer 0* and *detection layer 1*, as well as *detection layer 2*, which consists of a trained model for all individual features combined, enabling even more synergistic effects. When analyzing an unclassified request, all of those detectors are applied, such that each detector decides, whether the request is normal or anomalous. DUMONT classifies a request as anomalous, if at least one detector triggers an alarm. This approach provides a maximal detection rate, though increasing the false-positive rate as well. The alternative - classifying a request as anomalous when at least n detectors trigger alarms - is no option here, as it allows malware to cover its communication by hiding information within the features of $n - 1$ detectors.

2.3.3 Training DUMONT

For the training of DUMONT several steps are necessary. Initially the normal data is split into training, validation and testing datasets, maintaining the temporal order of the requests. After training and selecting a suitable model for each detector, a sample of malicious communication is used to calibrate the hierarchical detectors of DUMONT.

Model selection

Training a model of normality with a One-Class SVM requires optimizing two parameters, namely the width of the Gaussian kernel and the "softness" of the One-Class SVM. For each combination of these parameters a different SVM model is obtained. From those models a suitable one is picked by the following heuristic:

1. A threshold is set to define an upper bound on the desired false-positive rate on the training dataset. Of the calculated models the one with the highest false-positive rate below this threshold is selected.

2. If models with identical false-positive rates occur, the one with the highest number of support vectors is selected, as this corresponds to the best adaptation to the training data.

Automatic calibration

After selecting a suitable model for each detector on the normal training dataset, the representation quality of the model is further optimized by calibrating the radius of the soft margin of the SVM on the validation dataset using a Receiver Operating Characteristic (ROC) curve. To generate a ROC, the validation data and a sample of malicious requests is processed with DUMONT using different thresholds (radius). In principle, a good threshold corresponds to the point closest to $(0.0, 1.0)$ in the ROC. As a low false-positive rate is the primary objective, however, the use of this point is not recommended, as it corresponds to a one-to-one ratio of false-positive and detection rate. The focus of DUMONT has to be a low false-positive rate, because each of the detectors in Figure 2.3 is able to trigger an alarm if it detects an anomaly. As a result the individual false alarms of the detectors are accumulated, rendering it highly important to keep the false-positive rate of each detector low. Though this results in generally smaller detection rates as well, the negative impact on the overall detection rate is small, since the correctly classified anomalies of the individual detectors are accumulated as well.

To assure a low false-positive rate for each of the individual detectors, the method illustrated in Figure 2.4 is implemented. Thresholds corresponding to a suitable ratio of false-positive and detection rate are found at the ascending gradients before any local plateau of the ROC curve. They are retrieved by positioning a linear function in $(0.0, 1.0)$ and selecting the point which is closest to that function. Two examples of such points are depicted in Figure 2.4 as black dots. While both of them have a good ratio of false-positive and detection rate, only the left one, selected by a linear function with a higher gradient, yields a low false-positive rate. Consequently the linear function with the steepest gradient is selected to determine the threshold for each detector in our system. We chose this approach to allow for a direct influence on the calibration of the false-positive rate. Otherwise also ensemble approaches [Die00] would be a viable option to train these classifiers.

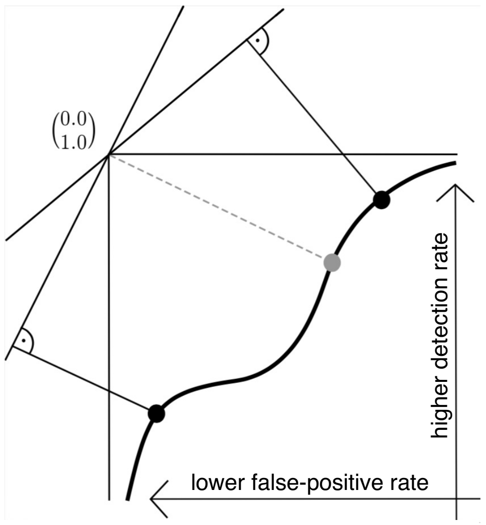


Figure 2.4: Calibrating a detector using the ROC curve.

2.3.4 Limiting Evasion

One problem of anomaly detection in general are evasion attacks that aim at poisoning the learning data [KL10]. If an adversary knows the distribution of the features of the normal requests, he can tune his malware to generate seemingly normal traffic with high anomaly scores. Such data points near the margin of the One-Class SVM can shift the model towards any direction desired by the attacker, resulting in a setup where once anomalous data is now classified as normal. Fortunately, different methods have been developed to increase the robustness of anomaly detection and to minimize the influence of an adversary. In particular, the techniques of bootstrapping [BS85] and sanitization [CSL$^+$08] can be applied to adjust and filter out anomalous data from the training corpus.

2.4 Empirical Evaluation

For the empirical evaluation, datasets of normal and malicious HTTP requests have been collected and used for different experiments. The datasets and the results of those experiments, as well as a comparison with a state-of-the-art approach, are illustrated in this section.

2.4.1 Evaluation Data

For collecting normal outbound HTTP requests, a dedicated proxy server has been set up at our institute. After discussing considerations of data privacy, six users were willing to use the proxy server for web access. In the resulting traffic dumps of outbound HTTP traffic, the IP addresses of all users have been pseudonymized.

Statistics of the resulting data set of 90 days is presented in Table 2.5. In total the six users generated 143MB of HTTP requests, consisting of 182,996 requests with altogether 173 days of usage and 5,272 requests per day.

	Number of Requests	Data volume	Active Days	Requests per active day
User 1	116,565	85 MB	50	2,331
User 2	29,723	36 MB	43	691
User 3	18,834	11 MB	52	362
User 4	9,882	6 MB	10	988
User 5	4,001	2 MB	8	500
User 6	3,991	3 MB	10	399
Total	182,996	143 MB	173	5,272

Table 2.5: Web traffic of six users recorded over 90 days.

For collecting malicious HTTP communication data, different sources have been used. In particular, samples of malicious software have been obtained from the Internet Early Warning system [EFG$^+$10] hosted at the University of Mannheim and different honeypots running at our institute. In total a dataset of 2.765 malicious executable files and PDF documents has been collected for our experiments.

To retrieve the kind of communication data relevant for our problem, a small virtual network has been set up, where the binaries and PDF documents are automatically executed in a virtual machine running Windows XP, providing each of them a time frame of 15 minutes to get active. The connections triggered by the malware are redirected to a virtual machine simulating the Internet using TRUMANBOX [Gor08]. To model the desired network layout, a HTTP proxy has been included in our setup. Any HTTP connections from the Windows machine had to find and use that proxy, using information found in the preferences and registry entries prepared on the Windows machine. Of the 2,765 malicious files only 695 have been capable of doing this, whereas several common malware families failed to correctly communicate with the web proxy and would not have been able to establish a communication outside a closed network.

Besides malicious software, there also exist public tools for establishing outbound communication channels to a system. In particular, the web backdoors MATAHARI[1] and RWW-SHELL[2] are included in the conducted experiments for creating covert communication. Both backdoors have been run with a polling interval of 10 seconds, executing 10–20 shell commands in each session. Moreover, the common tunnel software HTTPTUNNEL[3] is considered for tunneling various traffic through the web proxy.

It is noteworthy that we also executed additional experiments with other soft-

[1]A Simple Reverse HTTP Shell, http://matahari.sourceforge.net

[2]Placing Backdoors Through Firewalls, http://www.thc.org/releases.php

[3]GNU HTTP Tunnel, http://www.nocrew.org/software/httptunnel.html

	# Sessions	# Requests
Malicious software	695	12,899
HTTP tunnels	11	164
Web backdoors	12	345

Table 2.6: Statistics of malicious web traffic

ware for establishing tunneled communication, such as CORKSCREW, SKYPE and TEAMVIEWER. However, these tools make use of the `CONNECT` method for communication and have been excluded from our experiments, as they can be trivially detected and blocked. The statistics of the resulting dataset of malicious HTTP requests are presented in Table 2.6.

2.4.2 Evaluation Setup

To conduct the training of DUMONT in our experiments, the temporally first third of the normal data is selected. The trained models are then validated and calibrated on the temporally second third of the normal data and the validation partition of the malicious data. For realistic experimental results, the validation is repeated ten times, each time with a newly randomized set of validation data of the malicious dataset. For testing, the detectors are applied on the remaining temporally last third of the normal dataset, as well as the malicious test data, remaining in each of the randomizations. Due to that approach the final false-positive and detection rates are presented as the average values of those ten repetitions.

The features t_1–t_4 and t_* are not included in the evaluation. In practice they help detecting malicious outbound traffic at unusual times or with an unusual request frequency. Due to our methods of collecting malicious requests, however, this could not be fully tested, because for collecting malicious communication data the binaries and infected documents have been executed over night and weekend as well. Due to the resulting time stamps the corresponding time features contain artifacts and thus are easily distinguishable from benign communication.

2.4.3 System Performance

The detection and false-positive rates of DUMONT for each user are presented in Table 2.7. Applied on the traffic of tunnels, web backdoors and malicious software, DUMONT performs decently with detection rates of 100.0%, 94.3% and 89.3% respectively. The average false-positive rate reaches a value of 0.35%. While the detection rates of tunnels remain static among all users, the detection of backdoors and malware is strongly user-dependent due the variance of HTTP traffic. This variance is also the reason why rule-based methods are limited in detecting these covert channels, as we will see in Section 2.4.4.

In principle, each hierarchical detector contributes to the final detection performance of DUMONT. Covert communication can be spotted in all of the features

	Detection Rates				FP rates
	HTTP tunnels	HTTP backdoors	Malware	Malware (faked UA)	Benign web traffic
User 1	100.0 %	88.0 %	79.9 %	67.5 %	0.14 %
User 2	100.0 %	100.0 %	96.5 %	94.3 %	0.39 %
User 3	100.0 %	100.0 %	98.7 %	89.4 %	0.23 %
User 4	100.0 %	84.0 %	80.8 %	73.4 %	1.18 %
User 5	100.0 %	94.0 %	82.1 %	69.7 %	1.12 %
User 6	100.0 %	100.0 %	97.6 %	97.9 %	4.05 %
Average	100.0 %	94.3 %	89.3 %	82.0 %	0.35 %

Table 2.7: Detection performance of DUMONT

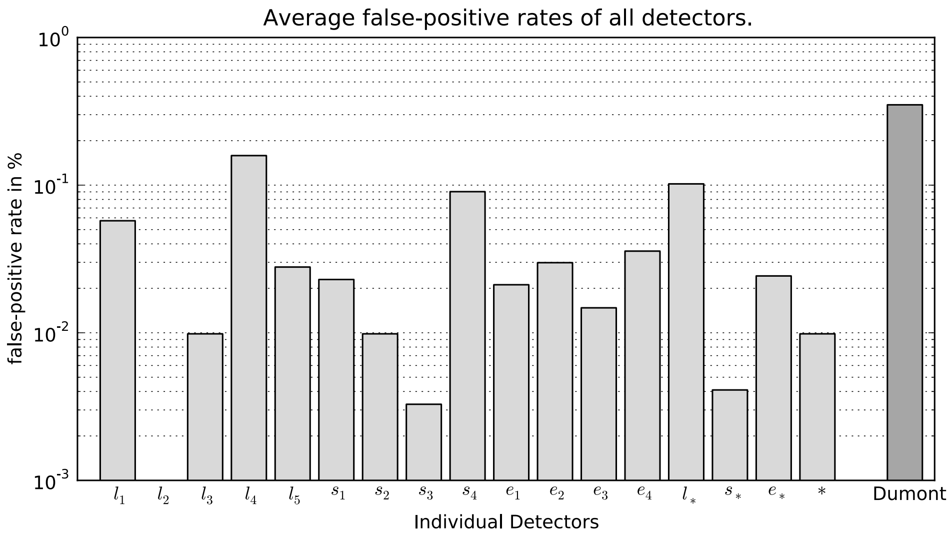
extracted from HTTP requests, as can be seen in Figure 2.5, which depicts the average contribution of the individual detectors on the false-positive and the true-positive rates. The false positives triggered by DUMONT are mainly caused by large data uploads and most notably cookies. While both types of requests are generally useful for interacting with the Internet, it is obvious that especially in our scenario they represent an inherent risk. Both methods are used to send a bigger and often encrypted amount of data to a server located outside of the protected network, which is what our system is designed to prevent. Such false positives could even be interpreted as true positives when found in a network with stricter security protocols. Since this is not our initial assumption, they are kept as false positives here.

DUMONT is implemented in Java, with no special performance optimization. On a single core of an Intel Core2 Duo with 3.00GHz, the whole normal dataset, containing the requests of six users of 90 days, can be processed (i.e. extracting the features and applying the trained detectors) within five minutes. This equates to a run-time performance of approximately 1.300 requests per second.

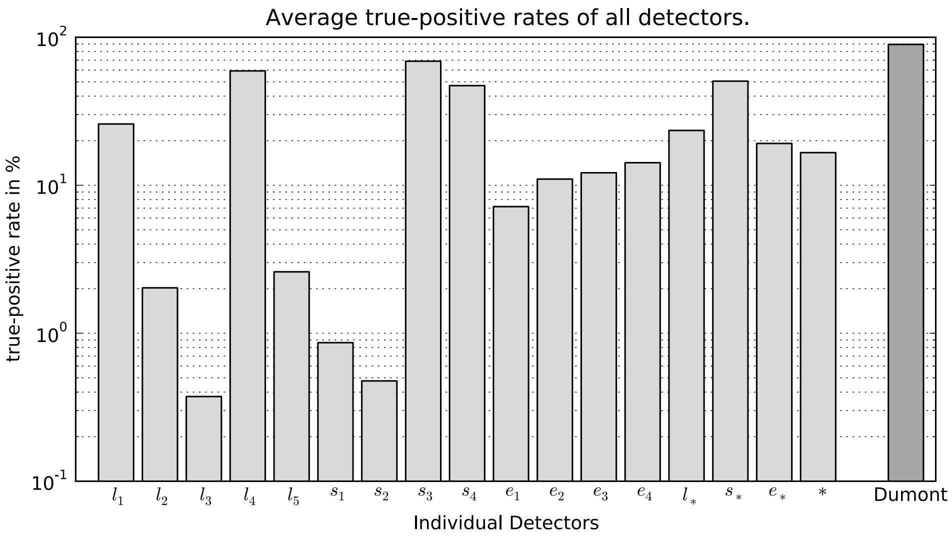
2.4.4 Comparative Evaluation

The second experiment conducts a comparison of the detection performance of DUMONT to WEBTAP [BP04], which detects covert communication using a mix of filters, trained rules and threshold values. The comparison is conducted on the dataset introduced in Section 2.4.1. In terms of the detection rate, WEBTAP identifies 100% of the traffic of tunnels, web backdoors and malicious software. As discussed in the previous section, DUMONT performs slightly worse. But in terms of false-positive rate, DUMONT significantly outperforms WEBTAP, which flags 3.6% of the requests as covert communication, thus generating more than ten times more false alarms in our experiments. These false alarms are due to the dynamics of HTTP traffic and can be attributed to changing header names and values.

To further illustrate this shortcoming, the influence of header changes on the



(a) Contribution to false positives.



(b) Contribution to true positives

Figure 2.5: Contribution of each detector to the detection performance.

overall detection performance is investigated. To this end, the *User-Agent* of each malicious request is changed to the one most frequently used by the tested user. As a result, all malicious requests contain ”faked” user agents. This method of masquerading malicious web traffic can be implemented into malware with only little effort. In this setting, the detection rate of WEBTAP drastically decreases from 100% to 3.7%. The performance of DUMONT, however, as presented in Table 2.7 decreases only slightly from 89% to 82%. Obviously, the detection of WEBTAP strongly depends on static header fields and thus can be easily thwarted. The comparison shows strikingly that already a little effort on the attackers side results in a security loss if the employed detection method can not be adapted to the individual characteristics and dynamics of HTTP communication.

2.5 Related Work and Limitations

Closest to our system DUMONT is the work of Borders and Prakash [BP04], which derive rule-based techniques for detecting covert communication in web traffic. While effective in different settings, these approaches assume that HTTP communication remains static over time and that features extracted from requests are stationary. With the adoption of HTTP as a generic communication protocol for many applications, these assumptions fail in practice and a more adaptive approach for modeling normality is needed.

The most common approach for detecting malware is the use of signatures. While in the past mainly focusing onto inbound traffic [Pax99, BBCP04, NKS05], recent work has studied detecting outbound malicious HTTP traffic via automatically generated signatures [RSL+10, RPF10, WBH+09]. In the case of a secured network, where malware tries to establish a covert outbound channel to leak specific information, an adversary surely will avoid using known patterns of malware communication and thus signature-based detection is not effective. Finally, various research on detecting the communication of bot networks [e.g. GZL08, GPZL08] is related to our approach and makes use of similar concepts. Yet this work focuses on identifying direct communication with end hosts and is not suitable for determining anomalous requests in a web proxy.

A different strain of research has studied techniques for circumventing the leakage of confidential data by monitoring sensitive data in host systems [SS11, KPPK10]. Although effective in practice, these approaches work only on a system where memory access can be monitored. By contrast, DUMONT can be directly deployed in a network without modifying the operating system of connecting hosts.

One of the *limitations* of detecting covert channels is based on general coding theory [McH95]. For HTTP, a good example is described by Feamster et.al. [FBH+02], where one party monitors accesses to certain benign web sites, while another party accesses those web site in a specifically arranged pattern. The transferred information is hidden within that pattern and therefore completely undetectable. Though fortunately such approaches limit the bandwidth of information to a minimum, they remain problematic, especially considering long term information leakage through insiders.

2.6 Conclusion

This chapter presented a novel approach for detecting covert and tunneled communication passing through a web proxy. Our system DUMONT builds on hierarchical detectors that can identify anomalous communication in various features of HTTP requests. By using machine learning techniques, DUMONT can be applied to the individual traffic of each user and thus can adapt to particular web surfing characteristics automatically. It is demonstrated empirically that this setting provides a better detection performance than current static approaches, where DUMONT can identify the communication of malicious software, tunnels and backdoors with only few false alarms.

An interesting topic for future work is to further extend the set of features. For example in a hybrid approach, features from keystrokes or mouse movement [ZP00] might be added to our system to achieve an improved detection performance. Similarly, daily bandwidth limitations as used in WEBTAP could easily be implemented to complement our approach. Finally, the integration of DUMONT into different network environments, e.g. for mobile devices or sensors, may provide perspectives for network-based detection of unknown malicious activity.

Chapter 3

Autonomous Learning for the Detection of JavaScript Attacks

Summary

Malicious JavaScript code in webpages is a pressing problem in the Internet. Classic security tools, such as anti-virus scanners, are hardly able to keep ahead of these attacks, as their obfuscation and complexity obstructs the manual generation of signatures. Recently, several methods have been proposed that combine JavaScript analysis with machine learning for automatically generating detection models. However, it is open how these methods can really operate autonomously and update detection models without manual intervention. This chapter presents an empirical study of a fully automated system for collecting, analyzing and detecting malicious JavaScript code. The system is evaluated on a dataset of 3.4 million benign and 8,282 malicious webpages, which has been collected in a completely automated manner over a period of 5 months. The results of our study are mixed: For manually verified data excellent detection rates up to 93% are achievable, yet for fully automated learning only 67% of the malicious code is identified. This chapter concludes with a discussion of the limiting factors, which would indeed enable a fully automated system, once they are solved.

3.1 Introduction

According to a study of Symantec [Sym11], the number of JavaScript attacks in the Internet has almost doubled in the year 2010, reaching peaks of over 35 million attacks per day. As part of these attacks, malicious JavaScript code is planted on webpages, such that a user visiting the webpage is automatically attacked and unnoticably infected with malicious software. The success of these attacks is rooted in the close interaction of the JavaScript interpreter with the web browser and its extensions. Often it is possible with a few lines of code to probe and exploit vulnerabilities in the browser environment [DHM08, EKK09].

Unfortunately, the detection of malicious JavaScript code is a challenging task: JavaScript attacks are small programs that are executed in the web browser. The attacker can build on the full flexibility of interpreted code, which allows him to easily obfuscate his code as well as dynamically exploit different types of vulnerabilities. Common security tools, such as anti-virus scanners, are hardly able to keep abreast of these attacks, as the obfuscation and complexity obstruct the manual generation of effective signatures. As a result, malicious JavaScript code is often insufficiently detected due to a lack of up-to-date signatures [RKD10].

As a remedy, several detection methods have been proposed that combine JavaScript analysis with techniques from the area of machine learning. These methods build on the ability of machine learning to automatically generate detection models from known samples of benign and malicious JavaScript code and thereby avoid the manual crafting of signatures. Common examples are the detection systems CUJO [RKD10], ZOZZLE [CLZS11] and ICESHIELD [HFH11], which are capable of accurately identifying malicious code in webpages at runtime with few false alarms.

Learning-based detection provides a promising ground for mitigating the threat of malicious webpages. However, to take effect and provide advantages over signature-based tools, learning-based methods need to operate with very little manual intervention. From the acquisition of training data to the generation of detection models, the learning process needs to be largely automatic to quickly adapt to the development of malicious software. Previous work has ignored this issue of automatic learning and it is open whether learning-based detection methods can really operate autonomously over a longer period of time.

This chapter tests the feasibility of automatic learning and presents an empirical study of a fully automated system based on the detector CUJO [RKD10]. The system (a) retrieves benign and malicious JavaScript code from the Internet, (b) identifies malicious functionality using client-based honeypots and (c) learns a detection model from features of static and dynamic analysis in regular intervals. The system is evaluated on a dataset of 3.4 million benign and 8,282 malicious webpages, which has been acquired over a period of 5 months. In particular the detection performance as well as the learning process over time are studied, for different features and learning methods, such as anomaly detection and classification approaches.

The results of our study are mixed: In line with previous work, the system attains a high detection rate of 93% if applied to manually verified data. However, in a fully automated setting it identifies only 67% of the malicious code in webpages - irrespective of the used features and learning methods. Two main factors that contribute to this decrease are identified:

- **Semantic gaps:** It is considerably hard to verify the presence of malicious activity during the visit of a webpage and use the exact same information at a later stage for learning. If both stages differ only slightly, malicious activity may be present but is not exposed to the learning method.
- **Time delays:** JavaScript attacks are very volatile and often active for only a few hours. Due to the large amount of processed data, a significant amount of time may pass between the verification of a malicious webpage and the resulting learning stage. If the malicious code is not present anymore, the detection model is trained on incomplete data.

The overall conclusion from this study is that fully automated systems for detection of JavaScript attacks are still an open problem and there exist several practical challenges that need to be addressed first.

The rest of Chapter 3 is structured as follows: The related work is discussed in Section 3.2. Section 3.3 then introduces our framework for data acquisition and presents details of the collected JavaScript code. Section 3.4 describes the features and learning methods used in our system. Section 3.5 presents the empirical results of our study and discusses their implications, which is continued in Section 3.6 with the analysis of different practically relevant training approaches. Finally Section 3.7 concludes this chapter.

3.2 Related Work

Before presenting the study on learning-based detection of malicious JavaScript code, some related work is reviewed first. In particular we discuss related approaches for analyzing and detecting malicious code in webpages. These approaches can be roughly categorized into *client-based honeypots*, *analysis systems* and *detection systems*, where these categories are not rigid and some systems implement a mixture of functionalities.

3.2.1 Client-based Honeypots

To systematically monitor and understand the phenomena of JavaScript attacks, several honeypot systems have been devised that visit webpages and mimic the behavior of users. One class of these systems are high-interaction honeypots, e.g. [WBJ$^+$06, PMRM08, SS06, Roa07], which operate a real browser in a sandbox environment and detect attacks by monitoring unusual state changes in the environment, such as modified system files. Another class of these systems are low-interaction honeypots, which only emulate the functionality of web browsers and corresponding vulnerabilities for tracking malicious activity, e.g. [IHF08, Naz09, BMB10].

Both types of honeypots are valuable sources for collecting JavaScript attacks, especially in combination with systems for efficient retrieval of potentially malicious webpages [IBC$^+$12]. In contrast to server-based approaches, client-based honeypots are capable of actively searching for malicious code and allow to capture instances of novel attack campaigns early on. As a consequence, client-based honeypots are widely used and can be considered a standard for monitoring JavaScript attacks in the wild.

3.2.2 Analysis Systems

Collecting malicious JavaScript code, however, is only a first step in crafting effective defenses. A second strain of research has thus focused on methods for automatically analyzing the collected code and extracting security-relevant information, such as patterns indicative for attacks. Most notable here is the community service WEPAWET that is backed by a chain of analysis tools for collecting, filtering and analyzing JavaScript code [CKV10, CCVK11, IBC$^+$12]. The service automatically analyzes webpages using an emulated browser environment and is able to identify anomalous behavior in the code using machine learning techniques.

In contrast to WEPAWET, which performs a more general analysis of webpage content, other systems address particular aspects of JavaScript attacks, e.g. [KLN$^+$11, KLZS12]. For example, the analysis system ROZZLE implements an involved multi-path execution for JavaScript code. Instead of following a single execution flow, the method inspects multiple branches of execution and thereby exposes hidden and conditional functionality of JavaScript attacks.

Although very effective in analyzing code and identifying JavaScript attacks, the presented analysis systems are mainly designed for offline application and induce an overhead which is prohibitive for real-time detection. For example, Cova et al. [CKV10] report an average processing time of 25 seconds per webpage for WEPAWET. For this reason, methods for offline analysis are not considered in our study - even if they employ learning-based components. Nevertheless, many of the techniques implemented for offline analysis are also applicable in online detection systems [KLZS12].

3.2.3 Attack-specific Detection

The first methods capable of detecting malicious code at run-time have been proposed for specific types of JavaScript attacks, e.g. [RLZ08, EWKK09]. These methods proceed by monitoring the browser environment for known indicators of certain attack types. For example, the system NOZZLE scans string objects for fragments of executable code, a typical indication of heap-spraying and other memory corruption attacks. While these approaches provide a low run-time, they are inherently limited to particular attacks and do not provide a generic protection from malicious JavaScript code. A more generic detection of JavaScript attacks is achieved by the systems BLADE [LYPL10] and ARROW [ZSSL11], which identify attacks using indicators outside the browser environment. In particular, BLADE spots and blocks the covert installation of malware as part of drive-by downloads, whereas ARROW generates detection patterns for the URLs involved in JavaScript attacks. Both methods intentionally do not analyze JavaScript code and are thus independent of specific attack types. However, by ignoring the actual attack code, these methods critically depend on the presence of the considered indicators in practice.

3.2.4 Learning-based Detection

The demand for a generic detection of malicious code has finally motivated the development of efficient learning-based detection systems, such as CUJO [RKD10], ZOZZLE [CLZS11], and ICESHIELD [HFH11], which are the main focus of our study. These systems analyze webpages at run-time and discriminate benign from malicious JavaScript code using machine learning techniques. In contrast to offline analysis, they induce only a minor run-time overhead and can be directly applied for protecting end user systems.

At the core of these learning-based approaches are two central concepts: the *considered features* and the *learning model* for detecting attacks. For example, ZOZZLE mainly extracts features from a static analysis of JavaScript code, whereas ICESHIELD monitors the execution of code dynamically and constructs behavioral features. Moreover, many efficient detection systems employ a supervised classification approach for learning, while the offline system WEPAWET successfully uses unsupervised anomaly detection for identifying attacks. We study these concepts and related differences in the conducted evaluation in more detail.

3.3 Data Acquisition

A key for evaluating learning-based detection systems is a realistic dataset of malicious and benign JavaScript code. Previous work has suggested to automatically acquire such data using client-based honeypots and offline analysis systems. This is clearly a promising approach, as it allows for automatically updating and retraining learning-based systems on a regular basis. However, almost no research has explored this approach in depth. Most of the results reported for learning-based detection have been obtained on a single dataset with manually cleansed training data.

This chapter investigates how learning-based systems perform if they are regularly and automatically updated with malicious and benign data *without* human sanitization. To this end we have devised a framework that visits malicious and benign webpages on a daily basis and returns reports for static and dynamic analysis of the contained JavaScript code.

3.3.1 Collection Framework

An overview of the collection framework is presented in Figure 3.1. The framework is constructed using existing security instruments, such as public services and client-based honeypots, and only serves the purpose of automatically retrieving large amounts of benign and malicious JavaScript code. Note that the framework is not designed to gain insights into the malware ecosystem. Moreover, learning-based components such as PROPHILER [CCVK11] and JSAND [CKV10] are deliberately excluded from the framework, as they may bias the evaluation of learning-based detection methods towards their specific feature sets.

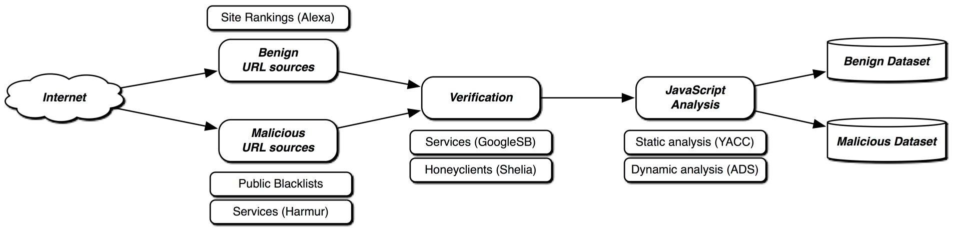


Figure 3.1: A framework for acquisition and analysis of JavaScript code.

Sources for URLs

At the start of each day sources of potentially benign and malicious URLs are harvested. For the benign URLs we considered rankings and listing of popular webpages. In particular, we randomly sampled 25,000 URLs per day from the Alexa ranking, which lists the top 1 million web pages according to visitors and page views. But popular web pages are not necessarily attack-free. In fact, attackers invest considerable effort into comprising popular webpages and exposing malicious code to a large group of users. Consequently, it cannot be ruled out that some of the 25,000 URLs are compromised, yet we assume that the vast majority of the URLs is benign. Moreover, we take precautions in the later verification to filter out known instances of JavaScript attacks.

For collecting potentially malicious URLs, we visited common blacklists and services tracking malicious URLs. As an example, the database service HARMUR [LC11] was queried for all malicious URLs that have been submitted in the last 24 hours. Furthermore, we regularly retrieve URLs from search engines using “dangerous” search terms. In total our framework collects about 8,000 potentially malicious URLs per day. Similarly to the benign sources, these URLs are not guaranteed to be malicious and thus the subsequent verification of the data is an indispensable step.

Verification

The collected benign and malicious URLs are far from being an ideal source for training and evaluating learning-based systems. First, the data is not guaranteed to be of a certain type and, second, even if a URL points to a malicious webpage, this does not necessarily indicate that JavaScript code is involved. Note that phishing and other scam webpages are also often flagged as malicious but do not contain any malicious JavaScript code. To improve the quality of our data, a verification stage is employed that filters the data, such that malicious and benign JavaScript code is obtained with high probability. For this task we prioritized tools that could also be applied in a practical deployment scenario, that is, the verification is conducted on a single workstation, using only techniques that return results within a few hours.

The following two verification procedures are applied to benign and malicious URLs, respectively:

(a) For improving the quality of benign URLs, blacklisted URLs are filtered out from the 25,000 URLs collected per day. To carry out this task efficiently, the Google Safe Browsing service is used, which provides a frequently updated list of malicious URLs. The main goal of this stage is to remove known malicious code from the benign dataset.

(b) All collected malicious URLs are analyzed using a high-interaction honeypot. In particular, the honeypot SHELIA [Roa07] is used, which monitors a target application and employs taint tracking for identifying memory corruption and code injection attacks. As target application the Internet Explorer 6 is used.

As a result of this analysis, the collected data can be refined to webpages that are either (a) likely benign and not contained in blacklists or (b) likely malicious and cause memory corruption or redirection of control flow during execution. We focus on a particular setting, namely SHELIA and the Internet Explorer 6, as this browser version is known for several public vulnerabilities and a frequent target of attacks. Nevertheless, our framework could be easily extended to also support other browsers and client-based honeypots for the verification stage.

Javascript Analysis

While learning-based detection systems generally follow the same design template, their inner working differs fundamentally. For example, ZOZZLE closely interacts with the JavaScript interpreter of the Internet Explorer, ICESHIELD uses frozen DOM objects for tracking JavaScript execution and CUJO makes use of a dedicated sandbox for analyzing code. Clearly, integrating all these different technical systems into a single prototype and conducting a fair comparison would be an intractable task.

Therefore we focus on a single system, namely CUJO, as it provides static and dynamic analysis of JavaScript code. While there are several subtle differences in how the code is analyzed, the reports of the static and dynamic analysis are basically similar to the representations of code used in the other systems. Overall, we are not interested in benchmarking concrete implementations but rather comparing underlying concepts, such as the use of static or dynamic code analysis for detection.

Static analysis. The static analysis in our framework first assembles the code base for a given URL by following redirects and downloading referenced JavaScript code. The assembled code is then parsed using a YACC grammar and the parsed representation is passed as a report for further analysis. A detailed description of the parsing process is presented by Rieck et al. [RKD10].

Dynamic analysis. Additionally, a dynamic analysis of the JavaScript code is conducted. This analysis uses an enhanced version of ADSANDBOX (ADS), an efficient sandbox for JavaScript code. This sandbox is embedded in the Internet Explorer and allows to observe the behavior of JavaScript code in a secure environment. All interactions of the code with the virtual browser are recorded and a detailed report of the code's behavior is generated. A description of the sandbox is provided by Dewald et al. [DHF10].

The presented framework enables us to automatically collect benign and malicious webpages and to generate analysis reports for each webpage on a daily basis. It is necessary to note that analyzing over 25,000 URLs and processing 8,000 URLs with a honeypot each day is a challenging task. As a consequence, several hours may pass during the processing of a webpage and a verified malicious URL may not necessarily expose malicious activity when it is later visited using the JavaScript analysis. This problem of *time delays* is inherent to our setting and would also exist in a practical application. Hence, the data used here is not artificially corrected, and inactive attacks are left in the malicious dataset.

3.3.2 Collected Data Sets

The framework for data acquisition has been deployed on April 19th 2011 at our site and collected malicious and benign JavaScript code for a period of 5 months (137 days). Though the data is not artificially corrected and inactive attacks are left in the malicious dataset, the reports of ADSANDBOX are filtered to remove those that are broken, empty, timed out during the analysis or are too small to show any specific behavior, since such reports corrupt the training process with their inconsistent features. Table 3.1 shows the total number of webpages before and after filtering, and the resulting dataset sizes.

As described in Section 3.3.1, initially 25,000 benign and 8,000 potentially malicious URLs have been collected per day. The applied verification significantly reduced the number of malicious URLs, as only a fraction of the candidate URLs has been capable to trigger an attack in our honeypot. Figure 3.2 depicts the number

	Benign	Malicious
Total number of URLs	3,400,000	8,282
Number of filtered URLs	2,900,000	3,220
Total size of dataset	359,000 MB	130 MB
Size of filtered dataset	291,000 MB	77 MB
Unique filtered URLs	1,200,000	2,146

Table 3.1: Details of the acquired datasets.

of malicious URLs that have been visited and verified per day.

The corresponding size of the downloaded JavaScript code per day for the benign and the malicious URLs, averaged for each day, is depicted in Figure 3.3. An interesting observation is the continuous growth of the amount of JavaScript code found at the benign URLs, reflecting the generally increasing importance of JavaScript. The peak at day 78 is due to a down-time of our analysis system and the resulting queue of unvisited webpages. For the malicious URLs a certain kind of recurring structure seems to exist, showing a weekly periodicity in the peaks and average lines. This finding can be credited to compromised workstation systems involved in the attacks, e.g. by redirecting traffic or hosting landing pages.

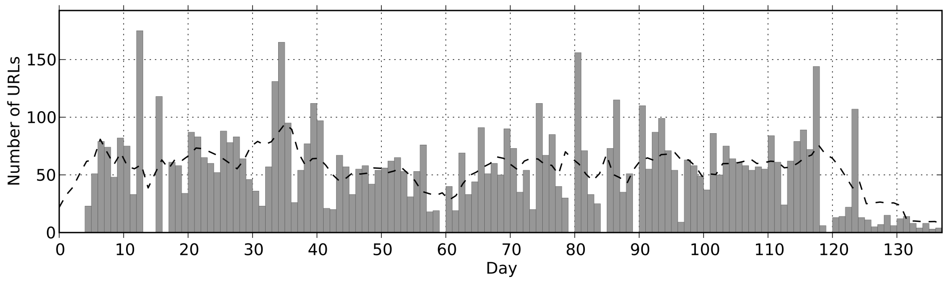


Figure 3.2: Number of malicious URLs visited per day. The dashed line represents a weekly average.

3.4 Learning-Based Detection

All learning methods heavily rely on the *features* chosen to model a certain problem. The features for the detection of malicious JavaScript code require the ability to reflect patterns in the code as well as monitored malicious behavior. Subsequently a proper *learning method* has to be selected, which is capable of handling those features. To overcome the limitation of choosing either a supervised or an unsupervised learning method, we discuss a way of evaluating both types of learning methods in a single setup. The following sections present this setup, the set of features and learning methods considered in our study, and the resulting learning framework.

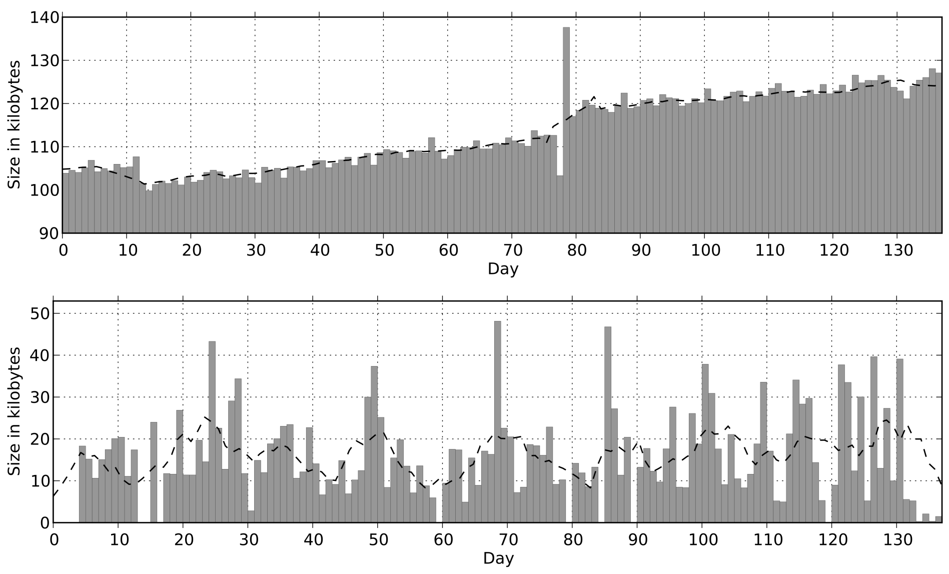


Figure 3.3: Average size of JavaScript code collected per day from benign (top) and malicious (bottom) URLs. The dashed line represents a weekly average.

3.4.1 Feature Extraction

We focus on features that can be extracted from the available static and dynamic reports. Those features are independent of specific attack characteristics, while reflecting relevant properties of the contained JavaScript behavior. In particular token n-gram-features are applied to extract short string features from the reports, similar to the q-gram features implemented in CUJO. Moreover, we use specific numeric features derived from WEPAWET and ICESHIELD.

Token n-gram features

Token n-grams denote sequences of n words. To extract token n-grams from a static or dynamic report, the report is transformed into a sequence of words, separated by white-space characters. The desired token n-grams are extracted by sliding a window with the size of n words over the report and storing the consecutive n words found at each position. The following example depicts this procedure for a snippet of a static report, using a value of $n = 3$:

`ID = ID + x` $\Rightarrow$ `{ (ID = ID), (= ID +), (ID + x) }.`

The procedure works similar for the dynamic reports, where the `SET` command is executed on a variable `x.y`, leading to a similar list of sequences:

`SET x.y to ”a”` $\Rightarrow$ `{ (SET x.y to), (x.y to ”a”) }.`

In the next step the token n-grams of each report are mapped into a vector space spanned by all possible token n-grams, i.e. each dimension is associated with the occurrences of one particular token n-gram. This results in a high-dimensional, yet sparse vector for each report, which can be efficiently processed [RKD10].

Numeric features

Additionally, 15 different numeric features are derived by analyzing the static and dynamic reports, similar to the way such features are extracted in WEPAWET and ICESHIELD. These features are designed bearing specific malicious JavaScript behavior in mind. Exemplary features inspired by WEPAWET are counts of the occurrences of `document.location` or `document.referrer`, the number of instantiated components, the number of times code is executed dynamically or the ratio of string definitions and string uses. Examples of features inspired by ICESHIELD are the number of occurrences of dynamically injected code, occurrences of potentially dangerous MIME-types or abnormally long strings used in decoding functions.

Unfortunately the features of WEPAWET and ICESHIELD could not be accurately re-produced, as some of them require information which is not contained in our analysis reports. Thus the combination of this incomplete feature set and our heterogeneous data leads to an insufficient detection performance for these numeric features. To avoid a misleading presentation of the results, numeric features are omitted in the following experiments.

3.4.2 Learning-Based Detectors

Taking insights from CUJO, the behavior of different learning models is studied using *Support Vector Machines* (SVM) [MMR$^+$01, SS02] with linear kernels. Though other approaches like decision trees work as well, SVMs offer a high performance and are robust against noise in the data. The feature space depends on the choice of the kernel. Focusing specifically on SVMs with a *linear kernel* has two advantages. The first one is the ability to process big datasets of very high dimensionality very fast. While Gaussian kernels often lead to better results than using a linear kernel, this comes at a massively increased cost of run-time and memory requirements. The second advantage is the opportunity to parametrically balance the influence of differently sized datasets during training. Before discussing this in more detail, some conceptual basics are covered.

Basics of two-class SVMs

A two-class SVM model is trained on datasets of two classes. Objective of the training is to find a hyperplane between the two classes which separates them with a maximal margin. The data points of each class with feature vectors closest to this margin are denoted as *support vectors*. In our setting, those two classes correspond to the reports retrieved from the benign and the malicious URLs. For each report the feature vectors are extracted. Once the model is trained on the feature vectors of those two datasets, an unknown report can be classified very fast. For this purpose the feature vector of this unknown report is extracted. Afterwards its position and distance from the hyperplane allows to predict the corresponding class membership. Figure 3.4 depicts an example of benign (white) and malicious (black) data points, where different models have been trained on. The dashed line of the middle image represents the separating hyperplane.

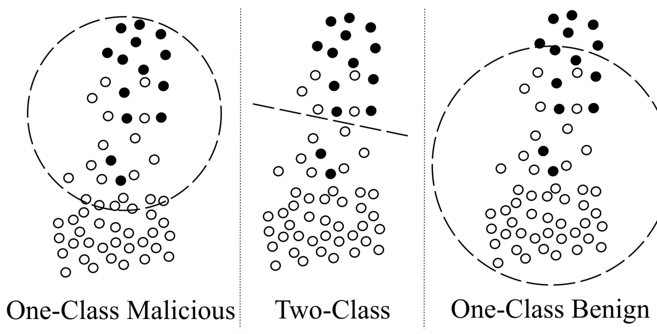


Figure 3.4: Visualization of different models achieved by different values of ω.

Two-class learning methods work best if the two classes have a comparable size. If one of the classes is much bigger, however, this approach often leads to suboptimal models. In that case one-class learning is often the better approach, allowing to learn a model on a huge dataset of one class alone.

Transition to One-Class SVMs

Two-class and one-class learning methods both have their advantages and shortcomings. Focusing only on unsupervised one-class methods means ignoring our available malicious data, which makes calibrating the detector much harder. Focussing only on the supervised two-class methods, however, leaves the problem of the imbalanced size of both datasets. To achieve a smooth transition between both methods, the weight parameter ω in included into the model selection phase of the learning method. A high value of ω increases the weight on one specific class, which means that the two-class SVM operates like a one-class SVM for this class. This is done by applying a higher penalty for misclassifications of this class. It is also possible to choose ω such that it balances two differently sized classes. An example of the models resulting in using three different values of ω is illustrated in Figure 3.4 on a toy data distribution.

The utilization of ω during model selection facilitates a direct comparison of the detection performance of a two-class learning method with both a benign one-class learner and a malicious one-class learner, simply by applying different values of ω during model selection. A hope is that a properly defined ω could lead to a model better adapted to the imbalance of the two classes. As a result, neither the one-class nor the two-class models would be expected to be the optimal solution, but instead an optimized model in between.

More details on the formulation of the optimization problem and the decision functions for the One Class and the Two Class SVMs, as well as their similarities and differences are provided in Chapter 7.1.

3.4.3 Learning Framework

The long-term dataset acquired during this study, as well as the different selected features and learning methods allow for an extensive experimental evaluation. This whole evaluation is conducted in an offline manner under utilization of the SVM library LibLinear [FCH+08] for training the model. The extraction of the token n-gram features from the static and the dynamic reports leads to a static and a dynamic detector, respectively. Taking insights from CUJO, the parameter n is fixed to $n = 4$ for the static reports and to $n = 3$ for the dynamic reports.

To evaluate the behavior of the different detectors during the whole time period of the study, the system is trained and tested weekly, i.e. every seven days all models are re-trained. To cope with the circumstance that the number of malicious reports is much smaller than the number of available benign reports, all past malicious reports are used during the training phase, but only the last two weeks of benign reports are used. This two week dataset is halved, such that the newest reports constitute the *training data*, while the older reports constitute the *validation data*. On these datasets a model selection for both the static and dynamic models is performed over the cost parameter c and the parameter ω. Then for each model the hyperplane is calibrated as follows: First the false positive rate of the current model is calculated on the validation data. Then the hyperplane is shifted such that a preselected false positive rate on the validation data, $FP_{val}(\Theta)$, defined by a threshold Θ, is not exceeded. Finally the model with the highest true positive rate on the validation data is selected.

Being able to define Θ is a vital component of training a learning based detector, suitable for the network environment at hand, because some environments simply require lower false positive rates than others. When testing the best model this targeted false positive rate is achieved with only minor deviations. Consequently no achieved false positive rates are shown in the evaluation, as they are close to the target values defined using the parameter Θ.

In our practical evaluation we pick a value of $FP_{val}(\Theta) = 0.001$ (i.e. 0.1%) as a good compromise of low false positive and high true positive rate. For some experiments, however, also $FP_{val}(\Theta) = 0.0001$ is considered to highlight specific properties. For convenience the Θ with $FP_{val}(\Theta) = 0.001$ is further on denoted as $\Theta_{0.001}$, and Θ with $FP_{val}(\Theta) = 0.0001$ is denoted as $\Theta_{0.0001}$. Note that the combination of the collection framework of Section 3.3.1 and this learning framework realizes a fully automated system for collecting, analyzing and detecting malicious JavaScript code.

3.5 General Performance Evaluation

The evaluation starts by investigating, whether learning-based detectors can easily be integrated in a completely automated tool-chain without manually sanitized data. This is empirically supported by this section, which provides a long-term evaluation of the performance of the different detectors and features utilized in our learning framework. First of all the detection performance of two commonly used anti-virus tools is determined on our datasets, which is compared to the performance of our learning framework. In the next step the performance of our learning framework on a sanitized dataset is shown. After that examples and reasons for misclassified reports and the influence of ω are discussed.

3.5.1 Performance of AV-Scanners

The two anti-virus tools AVIRA ANTIRVIR and AVG ANTIVIRUS have been tested on the original JavaScript code of our complete benign and malicious datasets. Both employ different analysis engines capable of detecting maliciously behaving JavaScript code. Table 3.2 shows the results of this analysis.

JavaScript Code		Avira Antivir	AVG Anti-Virus
Benign URLs	FP	0.0007	0.0003
Malicious URLs	TP	0.2760	0.3140

Table 3.2: False positive and true positive rates of two anti-virus tools.

The achieved true positive rates are quite low. This result comes as a surprise, considering the extensive initial verification steps of our system, and considering the circumstance that the anti-virus tools have been applied several months after the last day of data collection, which gave the anti-virus vendors enough time to update their signatures. The achieved false positive rates range between the targeted false positive rates of $FP_{val}(\Theta_{0.001})$ and $FP_{val}(\Theta_{0.0001})$. This permits an easy comparison of the corresponding true positive rates to those achieved by our detectors.

3.5.2 Performance of Detectors

Additional to the performance of the individual static and dynamic detectors, the performance of a disjunctive combination of both detectors is evaluated as well. This combined detector triggers an alarm if either the static or the dynamic detector does so. The resulting average performance values of the different detectors, obtained from weekly re-trained models tested on the following week, are listed in Table 3.3.

While the static and dynamic detectors are nearly on par, the combination of both is significantly higher. In comparison to the performance of the anti-virus

Detector	$\Theta_{0.0001}$	$\Theta_{0.001}$
Static	0.3525	0.5409
Dynamic	0.4931	0.6208
Combined	0.5451	0.6733

Table 3.3: Average detection performance of the different detectors.

tools, all detectors show a good detection performance. Especially the combined detector is able to classify approximately twice as much malicious URLs correctly. Surprisingly however, none of the considered features or learning methods attains a detection rate of more than 90%, as reported from previous work, which used manually sanitized datasets. To investigate this further we decide to create a subset of the malicious JavaScript code using the anti-virus tools as an additional sanitization instance. The assumption is that previous work always used manually sanitized datasets, so this way of automatic sanitization is expected to result in a better detection performance. In this new *AV-Alerts* dataset only those 890 URLs are included which both anti-virus tools raised an alert for. The results of the evaluation of our learning-based detection methods on this dataset are listed in Table 3.4.

Detector	$\Theta_{0.0001}$	$\Theta_{0.001}$
Static	0.7264	0.8446
Dynamic	0.7394	0.8480
Combined	0.8450	0.9319

Table 3.4: Average true positive rates of the different detectors, tested on the AV-alerts.

Especially the combined detector shows an impressively increased detection rate for both values of Θ, reaching a performance much closer to that of methods which are solely tested on manually sanitized data. This illustrates that – under the assumption of a sanitized dataset – the results of previous papers can be reproduced.

The application in a completely automated system, however, drastically decreases the detection performance. We see two main reasons for this unexpected behavior. While the verification phase of the collection framework acts properly in defining, whether the scripts of a potentially malicious URL really behave maliciously or not, we can not be certain that the specific malicious behavior SHELIA detected is also detected and reported by ADSANDBOX. Because the learned model builds on those reports of ADSANDBOX, it learns features that do not contain the initial malicious behavior any longer. We denote this as the *semantic gap*, because it is caused by the discrepancy of detection and learning mechanisms. The second reason is of a temporal kind. Because JavaScript attacks are very volatile and often active for only a few hours, any delay between the verification step and the creation of the reports may lead to a worse learned model, because the attack may already be inactive again and the malicious code not present anymore. We denote this as *time delays*.

3.5.3 Misclassification Analysis

There exist different reasons for misclassifications. One general reason is that the datasets are noisy, i.e. not all members of a class behave in accordance to their label. For the benign reports this means that, as explained in Section 3.3.1, the verification phase for the JavaScript code of benign URLs does not guarantee to exclude all malicious JavaScript code. Due to the *semantic gap* this is even more complex for the reports of malicious URLs. SHELIA might respond to a type of malicious behavior, which the static or dynamic reports of ADSandbox do not reflect. As a result the dynamic and static reports look benign. The dynamic reports might even be empty or have completely failed to execute. To test this last assumption, the malicious dataset has been further filtered, leaving only those 2,179 malicious URLs which did not result in errors in the dynamic execution. When testing the dynamic detectors on this dataset, the average detection performance increased only very slightly (approximately 1.5%), meaning that these errors in the dynamic execution do not influence the quality of the reports significantly.

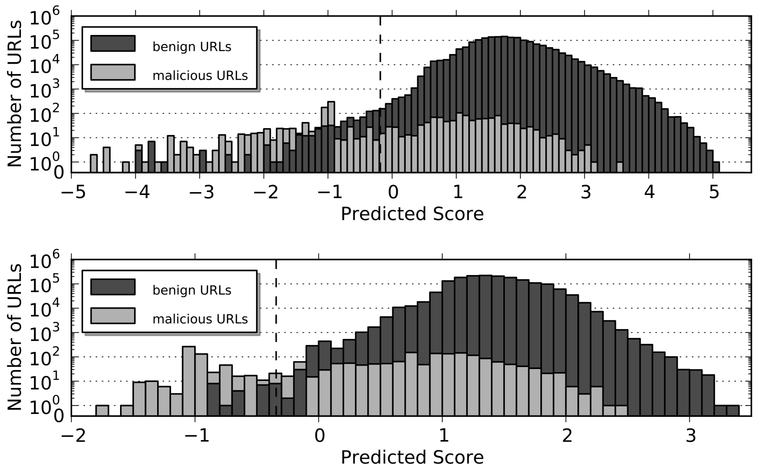


Figure 3.5: Histograms of the predicted scores of the optimized models of the static (top) and dynamic (bottom) detectors on the complete dataset. The dashed line represents $\Theta_{0.001}$.

To get a better idea of the concrete reasons for misclassifications in our system, we analyze the false positives and the false negatives that occurred using a representative model on the complete dataset. The histograms of the corresponding predicted scores of the static and the dynamic detectors, as well as the concrete values of $\Theta_{0.001}$, are illustrated in Figure 3.5. As a result it is found that many of the false positives actually are malicious. Concrete examples of code injection, the dynamic execution of long obfuscated sequences, redirects to suspicious websites, hidden iframes and heapspraying occurred. Those false positives that have really shown a benign behavior often contained features similar to malicious features, e.g. dynamic execution of obfuscated code or hidden iframes, which renders a proper classification difficult.

The verification process of the malicious URLs is more elaborated, especially due to the incorporation of SHELIA. Therefore none of the false negatives is expected to contain benign JavaScript code. For this reason it is an indicator of the *semantic gap* that many samples of the false negatives did not expose any behavior at all. Other misclassifications have been caused by malicious code looking very similar to benign code, i.e. omitting features usually found in malicious reports, like heapspraying, code injection or the dynamic execution of obfuscated code.

3.5.4 Performance of Different Learning Methods

Another interesting question is the impact of the class-weight parameter ω on the performance of the different detectors. Specifically, whether the detection performance of the normal two-class model can be optimized by this step, and how the different one-class approaches perform. The results are listed in Table 3.5. The low performance of the different one-class models does not come as a surprise. The learned model simply can not rely on one class only, and the calibration does not work that well either. The detection performance of the two-class model is boosted by the use of the optimized ω, resulting in much better results than those of the generic two-class model. The optimal models of our evaluation always use values of ω corresponding to a model in between a benign one-class model and a two-class model. An analysis of the development of the optimal values of ω during the re-training setup also shows a steady shift of ω towards the benign one-class model, caused by the continuously growing number of malicious data points.

	$\Theta_{0.001}$	
Learning Models	Static	Dynamic
One-Class Benign	0.0375	0.0303
One-Class Malicious	0.1314	0.1757
Two-Class	0.4560	0.4794
Two-Class Optimized	0.5409	0.6208

Table 3.5: Average true positive rates of the different models.

Integrating a class weighting into the training phase of a detector to balance differently sized datasets improves the overall performance. The insight that the chosen optimal ω resides closer to a benign one-class learning model is helpful as well, because the benign one-class detector is due to the stability of its database, i.e. the better availability of benign data, more desirable than a model trained on continuously changing malicious data.

3.6 Analysis of Re-Training Procedures

As this aspect greatly impacts the possibility to achieve a highly automated system in practice, we will now discuss the influence of different re-training procedures on the performance of the different detectors, aided by the previous long-term evaluation. The functionality and quantity of JavaScript code in real-life websites grows steadily, as we can see in Figure 3.3. A model should reflect this, because a model trained only once might be out-dated very soon. Thus a learning-based detector which is regularly re-trained on the latest datasets is assumed to achieve better models. The biggest disadvantage of this approach is the continuous requirement to regularly spend time and resources to compute an updated detector. While such learning efforts can be minimized, e.g. by using incremental learning [LGKM06], a confirmation of the assumed advantage of regular re-training in the domain at hand has yet to be done. For this purpose the following sections focus on a comparison of the long-term performance of the different detectors, either utilizing frequent re-training or conducting a one-time training only.

3.6.1 Regular Re-Training

In the re-training setup, the detectors are re-trained each week and tested on the following week. The corresponding overall average performance results have been discussed in Section 3.5.2. In the Figures 3.6 - 3.9 the weekly average is represented by the dashed line. The light vertical bars contained in the figures visualize days where for none of the 8,000 verified malicious URLs an actual malicious behavior could be exposed.

Figure 3.6 illustrates the performance values of the static detector. A first observation is the immense variance of both false and true positive rates. The picture is slightly different for the dynamic detector, illustrated in Figure 3.7. Its false positive rates show much less variance, while its true positive rates fluctuate massively, even more than the ones of the static detector. An especially interesting observation is, that the true positive rates of both the static and dynamic detector are weak in the beginning, while generally improving the average performance afterwards. This is caused by the low number of malicious data available for training at the very beginning of the evaluation. With the steadily increasing number of malicious reports available during training, however, better models are achieved. Another interesting observation is the performance drop of the models closely following day 78. At that day a huge amount of URLs has been updated after a down-time of the collection framework, which had an impact on the dynamic detector, but not the static detector.

The long-term performance of the combined detector, depicted in Figure 3.8, is more stable than the one of the individual detectors. Especially the periods of low performance of the dynamic detector are nicely backed up by the static detector. The general false positive rate has doubled as well, but the combination of the detectors still performs best.

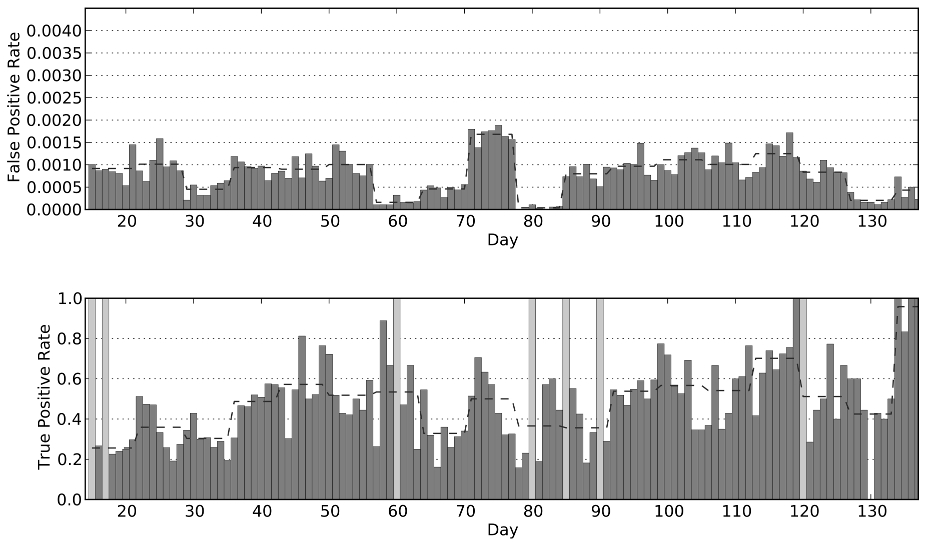


Figure 3.6: False (top) and true (bottom) positive rates of the static detector per day with regular re-training, using $\Theta_{0.001}$.

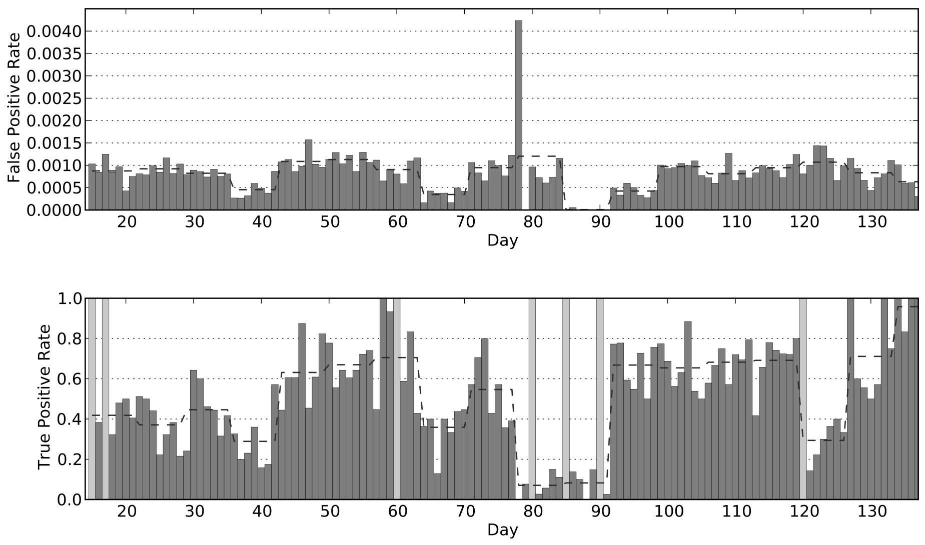


Figure 3.7: False (top) and true (bottom) positive rates of the dynamic detector per day with regular re-training, using $\Theta_{0.001}$.

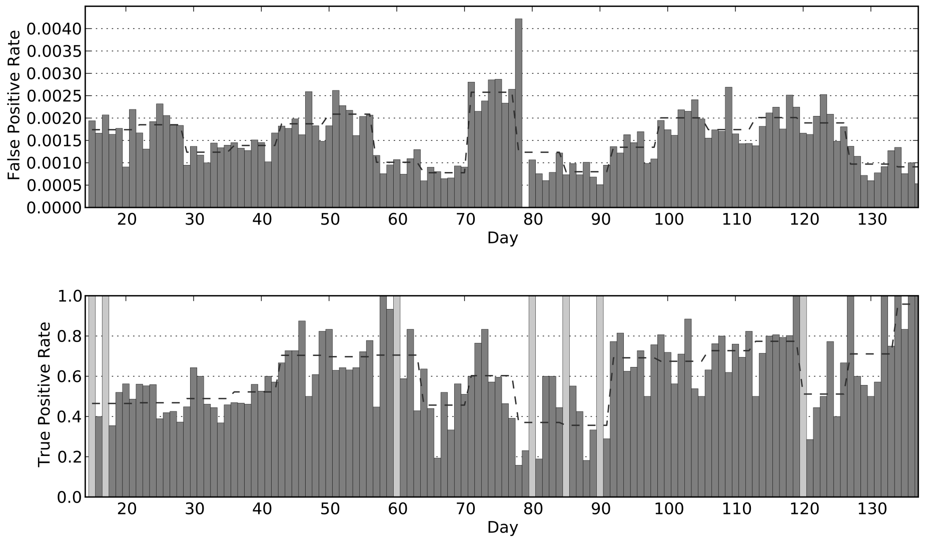


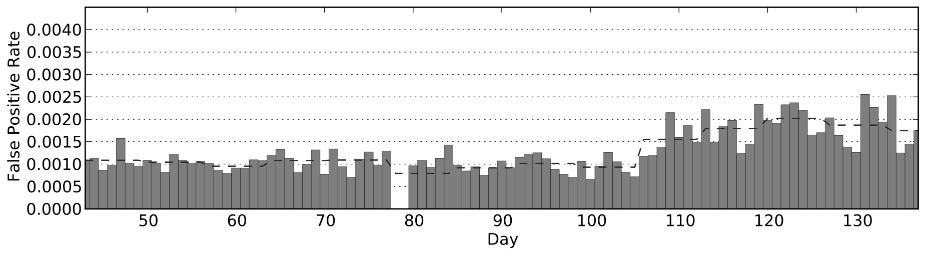
Figure 3.8: False (top) and true (bottom) positive rates of the combined detectors per day with regular re-training, using $\Theta_{0.001}$.

3.6.2 One-Time Training

In the one-time training setup a single model is trained on both the static and the dynamic reports, respectively. Because a sufficient amount of malicious data is vital for achieving a good detection performance, the model is not trained on the first weeks, but on the seventh, where a sufficient amount of malicious data is finally available. These models are then tested on all consecutive days. Note that for reasons of comparability the average detection performances of the previous tables are calculated on this range as well.

The dynamic detector exposes some interesting properties when comparing its re-training performance, depicted in Figure 3.7, with the corresponding one-time performance, depicted in Figure 3.9. The re-trained models show a much more varying performance than the model which has been trained only once. Especially the true positive rate is much more stable. The implication for the dynamic detector is, that a re-trained model is not necessarily better than an existing one. The static detector does not expose such varying behavior.

In terms of the long-term detection performance during the one-time training setup, an initial expectation was that applying an old model on newer data could lead to a steady performance decrease. This could not be observed for the static detector, where the false positive rate remained very stable and even the true positive rates follow closely the development curves observed in the re-training setup. The performance of the dynamic detector in Figure 3.9 does comply with that initial expectation a little more. The false positive rate, while stable for quite some time, starts to increase from day 100, and the true positive rate is below the average for



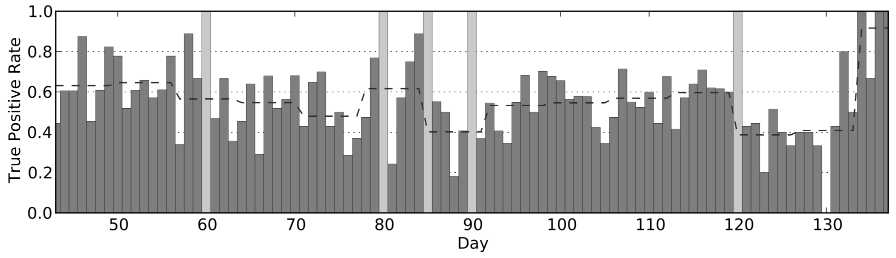


Figure 3.9: False (top) and true (bottom) positive rates of the dynamic detector per day, in the one-time training setup, using $\Theta_{0.001}$.

nearly two weeks at that time. These results suggest that using the detectors alone for such a long time without re-training is no viable option.

Detector	$\Theta_{0.0001}$	$\Theta_{0.001}$
Static	0.2444	0.5027
Dynamic	0.3371	0.5531

Table 3.6: Average true positive rates of the different detectors applied during one-time training.

Table 3.6 finally shows the average results during the evaluation of the one-time training period. The performance values are generally lower than those of the re-training experiments, especially for $\Theta_{0.0001}$ (see Table 3.3 for comparison). The implications for the static detector are that while the detection performance does not decrease or vary that much during the one-time training setup, its average performance is considerably lower than the one achieved with regularly re-trained models. The average performance of the dynamic detector also suffers, but as discussed above, the lower variance of the detection performance of the dynamic detector makes a less frequent re-training of this detector a viable option.

3.7 Conclusion

This chapter investigated the feasibility to combine a learning-based system for the detection of malicious JavaScript code with a completely automated system for the collection and analysis of JavaScript code. The behavior and detection performance of different learning-based detectors, based upon different features and learning methods, have been evaluated on a huge set of automatically collected data, where previous work solely used manually sanitized datasets.

The results of this evaluation have shown that the vision of a complete, automated, learning-based system has not been completely achieved. Two main factors have been identified as *limitations* to this approach and its results. The first reason is the *semantic gap* occurring when malicious activity is present during the visit and verification of a webpage but no longer during the subsequent learning stage. Already a slight discrepancy in this information transfer means that malicious activity may be present but is not exposed to the learning method. The second reason is the *time delay*, caused by the volatility of JavaScript attacks which are often active for only a few hours. Due to the large amount of data accumulated and processed during the visit and verification of the URLs, a significant amount of time may pass between the verification of a malicious webpage and the resulting learning stage. Consequently the malicious code may not be present anymore, which results in a sub-optimal detection model, because it is trained on incomplete data.

Fortunately these *limitations* can be overcome by creating an integrated system which combines the components of collection, analysis and detection very closely. As a result the complete behavior, exposed during the collection and verification of the malicious URLs, should be available to the learning component with a minimum time delay.

Furthermore the evaluation has shown that better models can be learned by integrating the class-specific weight ω in the model selection and taking care of a sufficient amount of malicious data. Combining different detectors is also important, because they often supplement each other. And finally it has been shown that a regular re-training mostly results in a better detection performance than using a single model for a longer time period.

To bring autonomous learning to reality, a critical step is the design and development of an integrated analysis and learning system. Besides that another important idea is the investigation of a more intelligent way of re-training the different models based on their comparative performance. For example one could re-train one detector regularly, but instead of relying completely on the newly trained detector, just use it in parallel to the current one. Also methods of online learning are an interesting option for that purpose.

Chapter 4

Classification of Structured Validation Data using Stateless and Stateful Features

Summary

To reliably identify problems impacting the service quality and system dependability of mobile communication networks, the monitored data needs to be validated. This chapter proposes and evaluates analysis methods, features and learning methods for the automatic validation of such data, with a special focus on failure data of mobile communication data. This data can be analyzed for discriminating failures caused by problems in the infrastructure (valid failures) from those caused by other circumstances like device imperfections (invalid failures), with the purpose of filtering the invalid failures, which effectively increases both dependability and value of the underlying data. To represent the complex structural and temporal properties of the mobile communication data, two complementary feature representations are proposed and compared, followed by a discussion of classification methods which are suitable for these feature spaces and for an interpretation of their results to support manual auditing. Their classification performances on these feature spaces are evaluated and compared to competitive approaches. In the evaluation a classification performances of up to 97% AUC-ROC is achieved. This renders our approach a good alternative to using manual matching rules, which require costly expert-knowledge and are much more time-consuming to define and maintain - while also highlighting the relevance of combining feature spaces of different problem perspectives. Additionally it is shown that using non-proprietary data analysis can enable feature representations nearly as expressive as those created by using proprietary analysis methods, which allows a broader application of the proposed methods, due to the lower processing requirements.

4.1 Introduction

The mobile telecommunication sector has grown rapidly in the last decades, with billions of mobile devices all around the world [CLC$^+$16] utilized today. Operating all those connections, in rural or urban areas, for static or fast-moving clients, requires a well developed telecommunication infrastructure. Optimizing the coverage of this infrastructure to reach as many people as possible, while maintaining the required quality of service, is an ongoing process, as users, locations and utilized protocols fluctuate. Therefore regular controls of the service quality data are needed to get an updated overview of the current state of the infrastructure, specifically on the most relevant locations, like densely populated urban areas. While the network service providers could in principle try gathering this kind of data directly on the customers devices, this is hindered by data privacy protection rules. Additionally the service providers are interested in measuring specific parameters like signal quality or bandwidth, which can only be derived via low-level access to the corresponding mobile devices. There might also be areas where the service quality is so low, that as a result data gathered from customers devices would simply not contain enough information to draw any useful conclusions.

For those reasons infrastructure data acquisition campaigns are conducted on a regular basis, either by the mobile service providers themselves or by third party contractors, which often provide analyses on this data, resulting in actionable results for the provider. Given the amount of time and money required for gathering this kind of data, a high reliability of its labels is indispensable to assure correct subsequent analyses. Failures, i.e. communication sequences which could not be successfully completed, are of specific relevance for the service quality and system dependability, as their analysis might point towards more severe problems in the infrastructure, e.g. coverage holes or insufficient interworking or malfunction of individual network elements on the involved protocol levels. Hence the detection and explanation of failures is highly relevant for the operator of the infrastructure, since they can potentially be solved once identified, before negatively affecting a large group of customers. For those reasons they are labeled as *valid failures*. On the contrary, sequences might also fail due to erroneous measurement environments or third-party server problems. Such *invalid failures* are not desired and need to be detected and removed from the data, as they are not relevant for the network provider, due to their relation to the measurement environment and other causes outside of its access. Discriminating those two types of failures is a crucial first step of data sanitization, resulting in a cleaner and better interpretable data foundation, as removing the invalid failures also strongly increases the dependability of the network system, since the valid failures provide a less noisy and more reliable data source for further analyses.

Current practical solutions for discriminating valid from invalid failures rely on manual analysis and labeling by domain experts, which is time consuming and costly. To accelerate and replace this manual analysis, we are proposing and evaluating different feature types and learning methods to achieve this objective with a semi-automatic approach. While the analyzed learning methods have been applied in the fields of NLP, process mining or sequence learning, they have not been applied to the classification problem of valid and invalid failures. Thus evaluating their classification performance on this specific problem, and discussing the practically relevant aspects is one part of the motivation of this chapter.

While the specificity of the classification problem does allow using prior knowledge to define handcrafted features for separating the valid from the invalid failures, their use is limited and expensive. For defining such features, the communication data needs to be analyzed to extract statistics and key performance indicators (KPI), which can then be used to find relevant properties. Besides the computational costs and the costs of manually defining such features through expert knowledge, this approach also produces features which are highly focused on specific data aspects, while potentially ignoring other factors in the traced data, which might additionally support the classification result. Therefore this chapter is also motivated by the need to evaluate the potential of applying a freely available communication data analysis method, instead of relying on proprietary analysis methods, which further reduces the cost (both in human and computational resources) of creating meaningful features. As the analyzed validation data offers both textual and sequential properties, specific *stateless* and *stateful* features are defined. which focus on these different data properties respectively. This allows an analysis of the nature of this type of data, by comparatively evaluating the effectiveness of each of those feature spaces individually, as well as combined. Consequently the results of this analysis are relevant for other fields, as they can be transferred to similar types of data.

Our approach is semi-automatic in that it initially requires manually labeled samples in the first phase of a new campaign. On this data classification models are trained, which can then be applied automatically on later sequences of the same campaign, supporting or replacing the otherwise required expensive manual analysis. This general processing is illustrated in Figure 4.1.

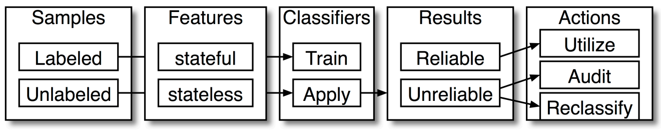


Figure 4.1: Main components of the processing chain.

Summarizing these motivational aspects, the analyses in Chapter 4 offer the following contributions:

- proposal of a system to semi-automatically solve learning problems on structured temporal data, applied to the use-case of the classification of mobile communication validation data

- comparative discussions of the required pre-processing, and evaluation of the potential of replacing expensive proprietary data pre-processing with computationally less expensive pre-processing, provided by freely available analysis and pre-processing tools

- proposal and analysis of suitable complementary stateless and stateful feature spaces for structured temporal data, which are evaluated comparatively, both individually and combined, using different classification methods commonly used by competitive approaches in related research fields
- additional discussions on the practical applicability of these feature spaces and classifiers, with a focus on system interpretability and reliability

Chapter 4 is structured as follows: This introduction is followed in Section 4.2 by a discussion of the related work. The datasets and feature spaces utilized are discussed in Sections 4.3 and 4.4, while Sections 4.5 and 4.6 discuss the compared learning methods and the consequences of their practical application. Section 4.7 competitively evaluates the classification performance of the proposed feature spaces, and Section 4.9 concludes this chapter.

4.2 Related Work

This section will discuss the related work and research, specifically the types and applications of stateless and stateful feature spaces for structured temporal data and token n-gram features. The learning methods utilized in related research fields will be introduced separately in Section 4.5, as they have to be discussed in the concrete context of the selected analysis methods and feature spaces.

4.2.1 Feature Spaces for Structured Temporal Data

Our main focus is to provide two complementary feature spaces obtained by pre-processing structured temporal data. We are talking about structured temporal data in the sense, that individual samples have to consist of a sequence of events, where each event is structured by and consisting of lexical and non-lexical tokens, where a token is a sequence of lexical and non-lexical characters. There exist various methods utilizing stateful and stateless data representations to solve problems on similar data. Stateful feature spaces are characterized through the way their data can be represented as a set of states and labeled or unlabeled transitions between those states. Exemplary types of this data representation are finite-state machines [Min67, Hop71], graphs [CSRL01, SB14] or event-based process representations [VDAADM$^+$11]. The flow of executed source code can also be represented this way, e.g. with call graphs as in [GDDC97, GYAR13]. Network communication data can also be represented in this manner, as it follows strict protocol definitions, which can be used to define such states and transitions. Once learned on a set of recorded network communication data, a stateful data representation can be used for different purposes, e.g. to learn and analyze the behavior of this specific network. Examples of this can be found in [CYLS07, CWKK09, CPWK06, CPC$^+$08, KGKR12, LJXZ08, NBFS06, WCKK08], where the authors used stateful data representations to learn, replay or simulate network behavior, and also in [CSS$^+$10, GWY$^+$15, LMD05, WBC10], where more specific communication patterns, like those of botnets or malware have been learned successfully.

Often such complex types of data representation are not required, especially when the discriminative properties required for the objective, e.g. multi-class classification, do not rely on state transitions. In that case stateless data representations offer a compelling solution. Instead of creating complex feature models, e.g. by explicitly encoding state transitions, single samples can easily be represented as individual feature vectors. Due to its lower complexity, stateless data representations allow a wider application of learning methods, while still achieving an often remarkable degree of representativeness of the information contained in the data, and correspondingly a high accuracy of the models trained upon it. Examples of stateless data representations and their use for training models of specific learning methods can be found in [CPWK06, BGH14, KKR11]. Both types of data representation can be retrieved from varying types of sequential or temporal data, e.g. from network traces like pcap dumps [II07], from programming language source code [RKD10, SBKR12], or from pre-processed data, e.g. analysis files which are the result of static or dynamic analysis, as for example explained in [DHF10, RTWH11]. Whereas the static analysis of raw data can be straight forward, e.g. by parsing lexical data into a direct textual representation, dynamic analysis is much more complex, involving the execution or replay of code blocks or network communication, allowing for a much more detailed analysis of the respective process flow. To reproduce state transitions, complex dynamic analysis is often required for a stateful representation of the data. However, it can also function as a good basis for a stateless feature representation. As a result, the feature space might be stateless - but its dynamically analyzed source data contains relevant information about the process states and its transitions. In [SBKR12] we showed for different types of network communication data and programming language code, that for solving classification problems, most often an initial static analysis is sufficient, while more complex data types require additional details of the process flow, only available through dynamic analysis. Finally the authors of [ABCM09] used a proximity-based feature extraction for log files, which works best on data with homogenous features with very discriminative positions, but is less suitable for our more heterogenous, less position-dependent data.

4.2.2 Token n-gram Features

When trying to achieve a stateless feature representation, the most obvious way is to project the lexical data into a vector representation. If there are numerical values contained in the data, and their range is known, one could start to parse those values and create numerical feature vectors based on them, e.g. as in [SR11]. If this is not the case, or if the data is more mixed, the more general n-gram feature representation can be used successfully. n-gram features are sequences of n consecutive characters or character sequences, like byte n-grams, consisting of n consecutive bytes, word n-grams, or token n-grams, where a token denotes an arbitrary byte-sequence. The basic idea is to see n-grams as a way to integrate local context into the description of the data to increase its information content. This type of feature representation has first been used in natural language processing [BPX+07, GSZ13, PSC15, WM12], even including the use of dynamic values of n lately, e.g. in [LXLZ15]. In recent years they have been applied to other fields as well, especially to those relying on lexical data representations similar to those of natural language processing, like the analysis of network communication data for IT security purposes. Some examples of research in this field are the works [Rie09, RTWH11], where n-gram feature spaces of natural language processing are applied to intrusion detection problems. In [ORLS14, PAF+09, WPS06, WS04] the authors use different types of n-grams to successfully analyze network communication for detecting server-side attacks or intrusion attempts, while in [RKD10, LŠ11] the authors use token n-grams for identifying malicious JavaScript code in web pages and even PDF documents. A similar approach of discussing the impact of n-gram properties of the dataset on the classification performance can be found in [WSAR13]. Although token n-gram feature spaces are very high-dimensional, the computing power of todays processors, coupled with modern learning methods like support vector machines [SPST+01, TD04] or neural networks [Bis95, MOM12] enabled using those feature spaces for solving complex classification tasks. Additional tricks further ease the use of those high dimensional feature spaces, e.g. using efficient data structures like Bloom filters [Blo70], using hashed feature vector representations [WDL+09] or hash kernels [SPD+09], or further reducing the dimensionality by applying additional sanitization steps during pre-processing.

4.3 Datasets

For our analysis we are using real-world mobile communication log datasets which are recorded an analysis of mobile network infrastructure coverage in Germany. As such the data is highly representative for and comparable with similarly collected network data. The data sets cover 22 consecutive days of data collected in the year 2014 and 25 days of 2015. The data was gathered by a fleet of cars, equipped with roof-mounted antennas and sets of multiple android smartphones. To better reproduce the geographical diversity, mobility and varying radio conditions of smartphone usage in practice, the cars are moving during the execution of those tasks. Each of the smartphones runs a dedicated measurement application which automatically executes different sequences of mobile communication. The resulting communication data is traced and stored in two formats. Pcap log files store the actual data transmissions, while Debug-logs store events of the participating smartphones, network events and also early dynamic analyses of additional properties like the success of network events like hand-shakes or transmissions. Each of the test sequences consists of a set of consecutively executed tasks. The nature of these service tests is to analyze the data service quality, as opposed to e.g. voice service quality. Therefore each of those tasks conducts a specific test, like connecting to a video-hosting website (e.g. www.youtube.com) and download a specific video, or connecting to a news website and visiting a number of articles. All of this requires the use of different protocols of the TCP/IP-stack, i.e. TCP, DNS and HTTP.

Based on this data one can conduct analyses of different properties of the tested mobile network, e.g. on aspects like cell coverage, signal strength or ping statistics. An important type of analysis is the sequence validation, i.e. the verification whether the sequence realized a successful communication. In practice this can be solved heuristically, e.g. by setting and requesting flags for crucial protocol flow positions, s.t. their completeness defines a successful sequence. However, sequences that have been verified to be unsuccessful are more relevant - and more problematic. The samples we are focusing on are of this problem class, here denoted as failures. Those failures can be analyzed to discriminate *valid* and *invalid* ones. Invalid failures can be caused by measurement errors like an erroneous test methodology or a malfunctioning packet sniffer, but also by factors outside of the access of both the measurement environment and the network provider. Examples for this are unavailable third-party servers which are required in test sequences, or unsupported protocol types like HTTPS. Because the measurement tools rely on specific events and triggers within the test sequences to continue their processing, HTTPS redirects are particularly problematic as they hide those events and triggers within their encrypted content. Valid failures on the other hand are those that are neither directly related to measurement problems nor to problems with third-party servers, and are therefore relevant for the infrastructure provider. Such failures range from application problems (e.g. failing replay of online videos), connection errors and server problems to problems with the protocol stack, like TCP or DNS problems, like unmatched pairs of SYN-ACK. Hence these failures are caused by the network infrastructure itself and are therefore of the highest interest for the network provider, because they point directly to weak points in their infrastructure. Equipped with that knowledge, the provider can fix them and thereby improve their network and communication service quality.

The concrete task to solve on this data is the classification of those valid and invalid failures. For our data set class labels are available. Those labels have been created manually by experts, which is a hard and time consuming task due to the variety of causes and the required expert knowledge. Fortunately the problem can be approached by using supervised classification methods, thus essentially labeling the data automatically, s.t. in the best case no further manual analysis is required, and in the worst case at least subsequently conducted manual analyses are eased and lead to more reliable results faster.

We are discussing two different types of data representation - debug log data and pcap log data - which need to be pre-processed independently. The debug-logs have the advantage of logging different types of events and are also containing results of early analyses processed during logging. As such they include many details which can not be easily obtained after the measurements are finished. A small disadvantage of this approach is that models learned on data collected this way are not easily applicable to data from other sources. We call the data created by using this analysis approach *debug* data, as its proprietary properties are closely related to reporting logs of debugging tools.

The format of the debug logs of the 2014 dataset is illustrated in the two exemplary events below, showing the format of the utilized sequence timestamp, message index, message timestamp and the data, consisting of a keyword in square brackets, and the actual message. The debug data of both years contains additional tokens, which add noise to the data. Those tokens (marked as underlined) are removed, as they are either redundant or do not contain information relevant for the classification task, but are instead based on information about the measurement itself, which could be learned by the model, and may lead to false assumptions, e.g. that a specific time range has a high probability for the occurrence of a certain class, which would hardly be transferrable to data of a different time period.

```
2014-07-23 13:02:02.484 0000312
MsgTime: 2014-07-23 13:02:02.484
 [MasterQueue] Removed cast:  8990956

2014-07-23 13:02:02.723 0000313
MsgTime: 2014-07-23 13:02:02.723
 [QOS EVAL] QOS-BASE: SUCCESS
 ([SMARTPHONE HTTP LIVE URL]
 TCP, DNS): Duration 710,00 ms
```

The structure of the data of the 2015 debug logs is overall similar to those of the 2014 campaign. However the occurring keywords have largely changed, as the utilized measurement tools and their output are under constant development. These fluctuations in the debug data format are a disadvantage in comparison to the tshark data, as it perturbs comparative analyses between different debug data sets. The following lines show examples of two single events of debug data of the 2015 campaign.

```
2015-02-05 16:06:51.764 0001169
 [TASK] (DEBUG) service:=FServices[i]
 Assigned(FSequence.FUsecase):  True;

2015-02-05 16:06:52.002 0001170
 [TRACING] (APP_EVENT) TloEvent:
 Received; TloSource:  SMARTPHONE HTTP;
 TloDestination:  EVALUATOR;
 Event:  Plugin ready; Plugin:  HTTP;
```

To achieve a standardized, more general data representation, the binary pcap-data is converted by applying TSHARK [1], a parametrically configurable tool for the analysis of network traffic traces, which is able to parse the contained TCP-stack communication, concatenate the individual frames and finally output a readable ASCII-file. We did not extract the hex-representation, the frame-wise details or the HTTP-payload, as these represent redundant information and are not required for a proper data description. Using the non-proprietary TSHARK for processing the network traces has the advantage of increasing the model compatibility to data collected by other methods, which eases a potential model transfer, at the cost of eventually missing some details, which are included in the proprietary and more dynamic analysis approach. We call the data created by using this analysis approach *tshark* data. The exemplary snippet below illustrates two events, i.e. two lines of a tshark log-file.

```
8319 137.194419
10.51.71.246 -> 173.194.44.95
 HTTP 542 GET / HTTP/1.1

8320 137.836234
173.194.44.95 -> 10.51.71.246
 TCP 68 80 -> 58780 [ACK] Ack=5
 Win=159 Len=0 TSval=1049 TSecr=1
```

We can see that this data format is very compressed, containing primarily the IP-addresses of both clients and the respective content, and does therefore not require removing additional fields. While both debug data and tshark data share an event-based, sequential data representation, their features differ. Due to its proprietary nature, the debug data contains many details about the execution of specific applications, the connection to certain websites and services, and even statistical details about the incoming and outgoing traffic. While the tshark-data contains significantly less detailed data, the availability of the required tools is its biggest advantage. As such, one of the objectives of this chapter is also to analyze, whether a

[1]tshark - the wireshark network analyzer

freely available analysis tool like TSHARK is able to produce analysis logs equally expressive as the proprietary debug-logs, which would enable its application to other fields that also allows access to pcap-data of its communication.

Data set	Data type	$\lvert S_{valid} \rvert$	$\lvert S_{invalid} \rvert$
2014	debug	4,550	393
	tshark	4,045	373
2015	debug	11,887	318
	tshark	11,743	316

Table 4.1: Sizes of the data sets of valid and invalid failures

Table 4.1 lists the number of samples of both classes of each dataset under utilization of these analysis approaches. The 2014 data offers less samples than the 2015 data. The differences in the numbers of samples between both datatypes of one dataset exist, because some samples were analyzable using the proprietary debug approach, while giving empty results when using the tshark approach, caused by the missing support for the analysis of specific network features in tshark, as well as the already mentioned deeper insights into the data, possible only with the debug approach. Due to these two different pre-processing approaches, the resulting log files differ in both their statistical properties and in their contained token sets. The implications of those differences for the classification problems are discussed in the following section about the resulting feature spaces.

4.4 Feature Spaces

This section introduces the feature extraction methods which define complementary stateless and stateful data representations to capture different data properties.

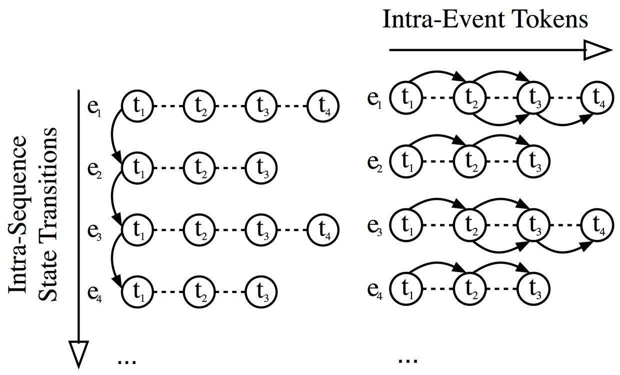


Figure 4.2: Conceptual example of the stateful $\Gamma_{s.ful}$ (left) and stateless $\Gamma_{s.less}$ (right) feature approach for 4 events of a structured sequence sample

Assume a sequence s consists of k events $e_1, e_2, ..., e_k$, where each event e consists of up to l tokens $t_1, t_2, ..., t_l$. Then the stateful feature space focuses on representing the state transitions of the process flow within the sequence (intra-sequence state transitions, i.e. the transitions between the events e) to finally discriminate both classes. The stateless approach is not aware of these transitions, but instead represents the sequences using a detailed static, high-dimensional token n-gram feature space (based on the intra-event tokens t) to model the data. Figure 4.2 illustrates these concepts. Using this approach creates two feature spaces which complement each other, as they rely on different data properties. Additionally the reliability of the failure classification process is increased by selecting interpretable feature spaces. Finally we will discuss differences between batch and online processing of the feature space, which is highly relevant for practical applications.

4.4.1 Stateless Features $\Gamma_{s.less}$

To achieve a high generalization, token n-grams are utilized for representing individual features of our samples. To describe the data in terms of token n-grams, each sample s is first represented as a list of lexical and non-lexical tokens. Moving a window of n tokens over this representation allows extracting all token n-grams of length n. We assume to have a dataset S, defined by its m individual samples $s_1, \ldots, s_m$. The function $\phi(s, n)$ extracts the unsorted token n-gram features of the single sample $s \in S$. The function $\Phi(S, n)$ agglomerates and sorts them to yield the feature space of the unified sets of token n-grams: $\Phi(S, n) = \bigcup_{j=1}^{m} \phi(s_j, n)$. Now we define $f_i \in \Phi(S, n)$ to be the ith dimension in the sorted token n-gram feature space. Based on that we can create a binary feature space. Using the final projection function $\Gamma_{s.less}(s, \Phi(S, n))$ we can project each sample s into the token n-gram feature space $\Phi(S, n)$, $\forall f_i \in \Phi(S, n)$, s.t.

$$\Gamma_{s.less}(s, \Phi(S, n)) = \begin{cases} 1, & \text{if } s \text{ contains } f_i \\ 0, & \text{otherwise.} \end{cases}$$

Exemplarily we can project a subsequence of a sample

$$\hat{s} = \texttt{'HTTP 542 GET HTTP/1.1'}$$

into the subset of a feature space $\hat{\Phi}(S, n)$ for $n = 2$ as follows:

$$\Gamma_{s.less}(\hat{s}, \hat{\Phi}(S, n)) = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix} \text{ for } \hat{\Phi}(S, 2) = \begin{pmatrix} \texttt{'HTTP 542'} \\ \texttt{'542 GET'} \\ \texttt{'GET HTTP/1.1'} \\ \texttt{'TCP 68'} \end{pmatrix}$$

Finally we normalize the feature vector by the vector sum of $\Gamma(s, \Phi(S, n))$, s.t. the vector norm equals one. This helps balancing longer and shorter samples. As a result we gain feature vectors for all samples $s \in S$, which have the same length of $|\Phi(S, n)|$, i.e. the number of token n-grams in $\Phi(S, n)$, because all of them are projected onto the same feature space $\Phi(S, n)$.

Before these token n-gram projections of both debug and tshark representations of the mobile communication data can be created, their individual token sets have to be extracted. To separate the individual tokens in the lines of the debug- and tshark logs, the following delimiters are selected: `()[]:;,␣*=."%<>`. Specifically the '=' is highly important, as it allows creating semantically relevant pairs of variable and value, which are then included in the corresponding token n-gram. To keep the notation simple, we are using from now on $\Phi(S)$ if the choice of n can be arbitrary, or simply Φ if the choice of n can be arbitrary, and the required dataset S can be derived from the respective context. After tokenizing the data we need to quantize strings tokens representing large numbers, also known as binning, i.e. grouping data into bins. It is used to enable discrete feature spaces and reduce their overall dimensionality. Without quantization, the overall number of token n-gram dimensions would fast become very large and computationally costly. Furthermore a string-comparison between numbers only matches if both are identical and not just metrically close. Therefore it is required to quantize different numerical fields in the log-files, especially the variable values in the tshark data and the relative timestamps of each task. In the tshark-data, the numerical values of the following parameters have been quantized (i.e. divided and converted into an Integer-value): the values of `Seq` (i.e. the sequence number) by 10, the values of `Ack` and `Win` by 100 and the values of `TSval` and `TSecr` by 1 million. This approach projects similar values into the same quantization bins, which allows using them as qualitative features. The quantization sizes have been selected s.t. a sufficient number of values is projected into the same bin. The tshark-data contains timestamps of the individually executed tasks as relative values, i.e. starting at zero for each new task. Using such relative timestamps (as opposed to absolute ones) enables the comparison of inter-task and inter-sample behavior. To increase the comparability, each of those relative timestamps is quantized as well, using a bin size of 1 second. Again the idea is to project temporarily close moments into the same bins, similar to a temporal sliding window, and thereby enabling their use as qualitative features. It may seem counterintuitive to restrict the existing numerical values by applying quantization. Actually it would be possible to extract those specific numerical values and learn separate numerical models on them, but this is neither the focus here, nor is it easily compatible with the use of token n-gram vectors.

Data set	$\lvert\Phi(S_{debug}, 1)\rvert$	$\lvert\Phi(S_{tshark}, 1)\rvert$
2014	138,940	199,577
2015	547,535	315,606

Table 4.2: Sizes of the Stateless Token Feature Spaces

Properly handling the size of the feature space to achieve a compromise between size and expressiveness is a major concern when constructing a feature space. Reducing the size of the feature space helps in achieving lower computational costs, but bears the risk of removing dimensions relevant for specific data set properties. The sizes of the resulting token feature spaces (i.e. $n = 1$) are listed in Table 4.2. The 2015 campaign contains much more tokens for both types of data representation. For the debug-data this is mostly due to the fact, that for the 2015 campaign a lot more keywords were introduced in the debug protocol. Additionally the individual test sequences of the 2015 campaign are also slightly longer, which increases the feature space sizes of both the debug and the tshark data representations. For our experiments we are using a token n-gram size of $n = 2$, because even n-gram sizes tend to represent pairs of parameters and values nicely, present and highly relevant in both types of data representation. To decide on a good value of n, the ratio $f_{pm}(\Phi(S, n))$ for the stateless feature space $\Phi(S, n)$ of a given data set S and a selected n is calculated, which is the ratio of *token n-grams with parameter-value pairs* to *all token n-grams* within this $\Phi(S, n)$:

$$f_{pm}(\Phi(S,n)) = \frac{\sum\limits_{\forall t \in \Phi(S,n)} f_{pm}(t)}{|\Phi(S,n)|}$$

with $f_{pm}(t)$ allowing to find adjacent parameter-value tokens as follows:

$$f_{pm}(t) = \begin{cases} 1 & \text{if } \exists i \in \{2, ..., n\} \text{ with} \\ & \hat{t}_{i-1} \notin \mathbb{R} \wedge \hat{t}_i \in \mathbb{R} \vee \hat{t}_{i-1} \in \mathbb{R} \wedge \hat{t}_i \notin \mathbb{R} \\ 0 & \text{else} \end{cases}$$

where $\{\hat{t_1}, ..., \hat{t}_n\}$ are the tokens in the token n-gram t. n-grams of size 2 also offer a good compromise between feature space size and expressiveness, which is especially important when combining the feature vectors, as large differences in their respective dimensionality would decrease their feature compatibility, reducing the performance of the classifiers trained on these combined feature vectors. The top plot in Figure 4.3 shows these respective values of the parameter-value ratios for the discussed data sets, for a range of relevant values of n. The bottom plot in Figure 4.3 additionally illustrates the sizes of the feature spaces for token n-grams with $n \in [1, ..., 5]$. We can see the values of the debug-data having a lower gradient than those of the tshark-data. This is caused by the higher n-gram variance in the tshark data, and specifically in the very heterogenous tshark 2015 dataset.

Using this approach to define $\Gamma_{s.less}$ also has some *limitations*. $\Gamma_{s.less}$ is defined by a single properly sized n for the definition of its token n-grams, which specifically aims for a high parameter-value ratio. As a result, a larger context defined by a token sequence larger than n might not be sufficiently represented. While using multiple or larger values of n would help alleviating this problem, it would also lower the influence of the parameter-value pairs, and is therefore less optimal. Further *limitations* are introduced by choosing fixed quantization values, which were chosen here as a compromise to allow both sufficient feature discrimi-

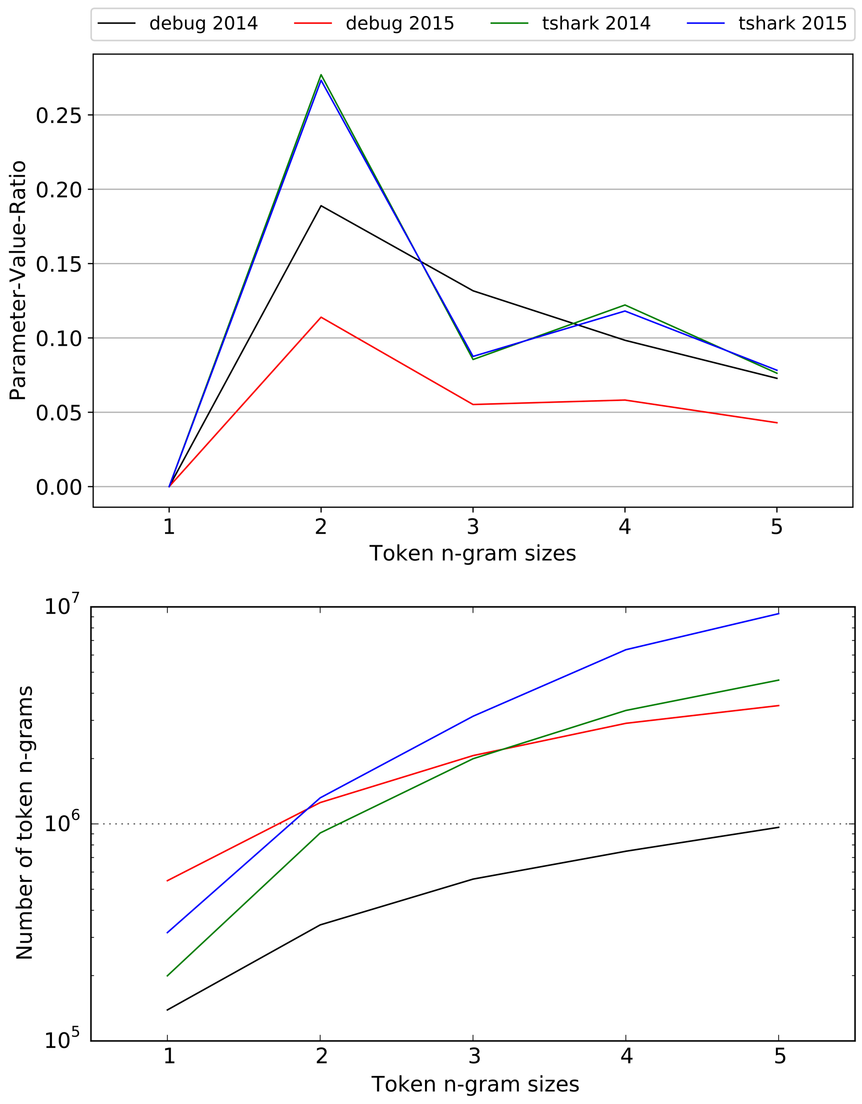


Figure 4.3: Stateless Parameter-Value ratios (top) and Feature Space Sizes (bottom) for the discussed data sets

nation and a sufficiently low feature space size, but may vary in other applications with different value distributions.

4.4.2 Stateful Features $\Gamma_{s.ful}$

In the stateless feature space all occurring token n-grams are used to define the feature space and project the samples. Now we want to focus on finding combinations of states and transitions reflecting the stateful behavior of each sample, to project the data sets into a stateful feature representation. The states should reflect measurements at consecutive moments and should also allow constituting similarity between similarly behaving samples. A shared property of the log-files is that each of their lines can be separated into their respective string tokens, and that each line represents a single, consecutive event (or state), temporally located by its timestamp. We use combinations of selected tokens shared between the samples to define those states, and their sample-wise order to represent the transitions of the underlying state-machine, resulting in a stateful representation of each sample. Note that this representation is similar to event-based process representations, as commonly used in process mining [VDAADM+11], which motivates the upcoming selection of learning methods used in this related research field to enable a comparative classification performance evaluation.

Dataset	$\lvert\Phi(S_{debug})\rvert$	$\lvert\Phi(S_{tshark})\rvert$
2014	45	3,181
2015	65	3,462

Table 4.3: Sizes of the Stateful Token Feature Spaces

Similar to the procedure for the stateless features, we denote a function $\Phi(S)$ to extract the token combinations relevant for representing the state-based semantics of each sample. For the tshark-data we start with the steps described in Section 4.3, i.e. tokens which are irrelevant for the stateful feature space, like timestamps or frame numbers, are removed, as long as the consecutive order of the log file entries is kept intact. Tokens that look most promising in terms of the protocol state-machine are the utilized protocols and ports. The IP-addresses are not so useful, as they are hardly generalizable due to their variation over different smartphones and campaigns, restricting the potential to compare data of different devices. However, the tokens representing the utilized protocols and ports (like `HTTP 540`) allow a detailed analysis of the state behavior within the individual sample, while being highly comparable between different samples and even across campaigns.

The approach for the debug data is similar. For the debug 2015 data there are up to four interesting tokens, which are used by the debug-system itself to describe its different states. These are hierarchical tokens like: `[ADB], [APP_EVENT], [TloEvent], [Received]`. The debug 2014 data does not use such hierarchical debugging keywords, but instead uses single, non-hierarchical keywords like `[COMMAND], [DASHBOARD]` or `[SYSTEM_LOAD]`. The resulting feature space sizes of both debug and tshark and debug data are shown in Table 4.3.

In contrast to the $\Gamma_{s.less}$ features, the token combinations collected here by $\Phi(S)$ can not be directly used to create feature representations, as they would have different lengths for all samples, which is based on the number of events per sample. However, feature vectors of identical length are a necessary requirement to allow processing by subsequently applied learning methods. We will use $\Phi(S)$ to create the $\Gamma_{s.ful}(s, \Phi(S))$ feature vector representations to achieve this goal. As this strongly depends on the selected learning method, we will use Section 4.5.2 to further discuss this topic.

By restricting the set of tokens utilized to define the states, this approach of $\Gamma_{s.ful}$ obviously has its *limitations*. One limitation is that such an approach prefers data sets with more tokens and more possible token combinations for defining states, which allow a more discriminative description of the occurring transitions. Furthermore this approach also assumes that the respective state-based behavior of the process state-machine is sufficiently reflected by these tokens, which may pose to be a problem in practice, if the utilized analysis tools do not provide the required analysis depth to achieve such a detailed description.

4.5 Learning Methods

The choice of learning methods is largely constrained by two factors. The learning methods need to be able to *(a)* perform well on our feature spaces to be suitable for the task, with primary objectives being reasonable computational costs and scalability with $|S|$ to perform well in a practical application, and *(b)* also need to provide interpretable results to further increase the dependability of the model, s.t. they can be used in a processing chain with a manual analysis step in post-processing [BBM+15, BSH+10]. If there are only unlabeled training samples available, past research [II07] has shown that unsupervised learning methods like one class support vector machines [SWS+00, MMR+01] can work well on token n-gram feature spaces [WSAR13] of abstract source code feature spaces [YWGR13]. Once labels are available, however, better results are achieved using supervised classification methods like Support Vector Machines [CV95, STC04] or Neural Networks [LXLZ15], especially for network communication data, e.g. in the domain of IT security [WSAR13]. For this reason they should be preferred if the sample labels are available. If the samples are only partially labeled, approaches like active learning [TK02] and the close relation of specific supervised and unsupervised learning methods, as elaborated in [SBKR12] open interesting approaches for using both methods complementary. In terms of interpretability, the Support Vector Machines also have the advantage of allowing nice interpretations of the model and classification results [HMG+14]. Due to those properties, their overall good performance and the availability of labeled samples of the individual valid and invalid failures, Support Vector Machines will be our primary learning method. More details on the optimization problem, the decision function and the use of kernels for Support Vector Machines are provided in Chapter 7.1.1.

For a comparison with the classification performances with other relevant and commonly used learning methods, we are additionally evaluating a Multi Layer Perceptron Classifier [PVG^{+}11] on both the stateless and the stateful feature spaces. MLP are widely used in process mining [MLI^{+}15, LWR^{+}10], which makes its classification performance relevant, as the stateful feature representation is similar to the event-based process representation widely used in process mining. But this method is equally interesting for the performance comparison using the stateless features, as it originates from the field of NLP sequence classification [Col02]. Due to its non-linear and non-convex loss function the interpretability of its results is harder than that of the SVM though, which is why we do not focus stronger on this classification method. More details on the definition and application of MLP classifiers are provided in Chapter 7.1.3.

4.5.1 Learning Methods on $\Gamma_{s.less}$ Features

Kernel methods are known to perform very well in supervised scenarios (e.g. [CV95, STC04]). We use a linear kernel, since research in [YHL12] and also our own research has shown that using a linear kernel with qualitative features yields good results at low computational cost, if the feature space is sufficiently large and the feature representation of each sample is sufficiently non-sparse. Both of those conditions are met here. Thus a model of a two class SVM is learned on the feature vectors of a subset of the training data, $\Gamma_{s.less}(S_{train})$. Since it is a supervised learning method, labeled samples of both relevant classes are required. The objective of the training is to find a hyperplane, separating the classes pairwise with a maximal margin. The hyperplane can be described by its normal w and its distance from the coordinate origin, denoted by a bias term b. The samples of each class defining the borders of this margin are denoted as support vectors. Once the model is trained on $\Gamma_{s.less}(S_{train})$, a new sample s is easily classifiable by calculating its direction and distance from the hyperplane via a prediction function, s.t. s belongs to the positively labeled class if $\langle w, s\rangle + b > 0$ and to the negatively labeled class if $\langle w, s\rangle + b < 0$. Further details of support vector machines can be found in Chapter 7, as well as in [MMR^{+}01]. On the stateless feature space we are additionally evaluating a Bayes classifier, commonly used in process mining [PSBDL18], which predicts the most probable class affiliation based on the scalar product of the respective feature vector with the class-wise term-frequencies.

4.5.2 Learning Methods on $\Gamma_{s.ful}$ Features

The evaluation of the failure classification on sample projections into a stateful feature space $\Gamma_{s.ful}$ requires several processing steps to achieve feature vectors of equivalent length, enabling the application of support vector machines. After indexing the defined states an index for each line of the log-file is obtained, resulting in a feature vector. To yield feature vectors of identical length, we can interpret the sequences as directed graphs, with the state tokens defining the nodes ν, connected by transition edges defined by their order. This allows us to create an adjacency matrix A_u in an unsupervised manner, i.e. without concern about the label of the individual log-file, resulting in a feature space capable of representing both classes. A_u contains the probabilities of the transitions between the state nodes. E.g. the nodes ν_i of a state-space feature vector of $v = [\nu_2, \nu_4, \nu_3, \nu_1]$ correspond to the following consecutive transitions: $[\nu_2 \rightarrow \nu_4, \nu_4 \rightarrow \nu_3, \nu_3 \rightarrow \nu_1]$. At this point it is also possible to create adjacency matrices A_s in a supervised manner, i.e. one matrix for each class, which can be used to define a classifier based on the highest class path probability in the first order Markov chain. We use this classifier as a comparison evaluation method for the stateful feature space, which is especially relevant, as Markov Classifiers are commonly used in the research field of process mining [PNC11, LGN12, LSD+15, ULD16, LCDF+15].

To be able to apply a support vector machine on this feature representation, sparse sample-wise feature vectors operating on the same feature space are required, i.e. the feature vectors need to be of identical length. The current stateful representation of the samples based on their state transitions does not fulfill this requirement. However, each sample can be projected onto the unsupervised adjacency matrix A_u, resulting in a sparse representation containing only the transitions of this sample. The resulting matrix can be vectorized to finally yield equivalently sized feature vectors. When comparing the classification performance of an SVM with a non-linear kernel, operating on feature vectors of concrete transition probabilities of A_u, with that of an SVM with a linear kernel, operating on vectors of binary values for each occurring transition within A_u, the linear kernel SVM consistently performs better, which is why we use the latter for our subsequent experiments.

4.5.3 Learning Methods on Γ_{comb} Features

Since both proposed feature spaces focus on different data properties, their combined expressiveness is highly interesting. For that purpose both qualitative $\Gamma_{s.less}$ and $\Gamma_{s.ful}$ feature vectors of each sample are concatenated, yielding a *combined* feature vector $\Gamma_{comb}(s)$, $\forall s \in S$. This representation is then evaluated with the SVM and MLP methods, as described above. The idea is that both feature vectors should complement each other, giving a classifier trained on them the opportunity to optimize for feature combinations between both of them, finally achieving a higher or more robust classification performance than classifiers on each of the individual feature vectors alone. We tested this second approach by using an ensemble approach [Die00] of combining the results of classifiers of both $\Gamma_{s.less}$ and $\Gamma_{s.ful}$ feature vectors disjunctively. As this did not perform competitively at all, the respective results are not included in the evaluation.

4.6 Details for the Practical Application

This section addresses topics which are relevant to achieve a working practical application and the required level of acceptance for the proposed system, namely the differences in two re-training approaches for the system, and the possibilities of interpreting the final classification results with respect to the most important features.

4.6.1 Batch vs. Online Classification

While the classification approach proposed here could also be extended to inter-campaign classification (i.e. training on the data of one campaign to predict samples of another campaign), this introduces new problem like partially incompatible feature sets due to model shift and concept drift [SK12, VBMKM09]. As a result we are only evaluating single corpus, single campaign classification problems, in which a single corpus of logged and pre-processed data is used for both training and testing of classification models. To obtain robustness of the classifier, it is crucial that the campaign-specific parameters utilized during the data logging process of recording the network traffic, as well as the parameters for the processing of the resulting log files have not been changed during this process. Those parameters could include logging different IP layers, changing the degree of details, or adding or removing specific parameters of the logging environment. Under this premise we can assume a high inter-similarity for the samples of the corpus. The task is now to train a classification model over feature projections of a data set of labeled samples, S_{train}, to predict the labels of a set of samples S_{test}. In practice this problem can be approached with a batch or an online methodology, the choice of which depends on the available computational resources and the time constraints, i.e. how fast a predicted label for a new sample is required.

```
Data: $S_{train}, S_{test}$
Result: Classification Scores of $S_{test}$
for $S_{train}$, $S_{test}$ do
    Create $\Phi = \Phi(S_{train})$
    Create $\Gamma(s, \Phi), \forall s \in \{S_{train}, S_{test}\}$
    Train $M$ on all $\Gamma(s, \Phi), \forall s \in S_{train}$
    Classify $\Gamma(s, \Phi)$ with $M, \forall s \in S_{test}$
end
```

Algorithm 1: Batch Classification Algorithm

The single-pass batch-approach, described in Algorithm 1, creates the feature space Φ and its projections $\Gamma \in \{\Gamma_{s.less}, \Gamma_{s.ful}\}$ over the complete set of S_{train}, trains the classification model M on the projection S_{train}, and finally allows for classification of the projections of S_{test}. This differs in some major aspects from the online approach, shown in Algorithm 2. Here we assume not to have S_{train}

and S_{test} available instantly, but instead receive subsets of S_{train} and S_{test} consecutively, which we will denote as $\hat{S}_{train}$ and $\hat{S}_{test}$. While this sounds cumbersome, it is practical reality: while the campaign data is collected, new samples arrive all the time. While one could wait until all data is available and a manually labeled sub set S_{train} is created, this is not done in practice. Instead continuously new sets of samples are received, of which some are manually labeled ($\hat{S}_{train}$) and others are unlabeled ($\hat{S}_{test}$). The online approach is designed to allow to directly incorporate these new $s_{train} \in \hat{S}_{train}$ into the model, while also allowing to directly classify the new $s_{test} \in \hat{S}_{test}$ based on the feature space defined so far. As a result the feature space over S_{train} can not be created once in a single pass, but a more dynamic multi-pass approach has to be used, where for each new set $\hat{S}_{train}$ the feature space and the projections are recalculated, and the model retrained. This enables the required optimization process to take the dimensions of all labeled samples S_{train} into account. Once this is done, the new $s_{test} \in \hat{S}_{test}$ can be classified to get a prediction as fast as possible. Since later feature spaces Φ are based on a larger set of S_{train}, their resulting models become better and better. Therefore it is reasonable to repeat the classification, even for already classified samples of S_{test}, to increase the classification accuracy.

```
Data: $\hat{S}_{train} \subseteq S_{train}, \hat{S}_{test} \subseteq S_{test}$
Result: Classification Scores of $s_{test} \in S_{test}$
$S_{train} = \emptyset$, $S_{test} = \emptyset$
foreach $\hat{S}_{train}, \hat{S}_{test}$ do
    $S_{train} = S_{train} \cup \hat{S}_{train}$
    $S_{test} = S_{test} \cup \hat{S}_{test}$
    Create $\Phi = \Phi(S_{train})$
    Create $\Gamma(s, \Phi)$, $\forall s \in \{S_{train}, \hat{S}_{test}\}$
    Train $M$ on all $\Gamma(s)$, $\forall s \in S_{train}$
    Classify $\Gamma(s)$ with $M$, $\forall s \in \hat{S}_{test}$
end
```

Algorithm 2: Online Classification Algorithm

Each of those approaches has advantages, but also introduces some *limitations*. Whereas the online approach allows the fastest classification of new samples, it requires re-defining the feature space with each new $\hat{S}_{train}$, which results in a higher computational cost and a slighty lower classification accuracy. If the practical application allows for a certain time buffer, then the batch approach is more suitable. It requires collecting all available labeled samples before being able to classify S_{test}, has a lower computational cost as a result, and has also a better classification accuracy. Additionally the model optimization in the batch approach tries to optimize a larger number of dimensions than the respective optimization in the online approach - at least, until the last labeled $\hat{S}_{train}$ has been processed. This leads to the batch model being more susceptible to over-generalization, while the online model is more susceptible to over-fitting. In fact the batch approach is a special case of the online approach, with $\hat{S}_{train} = S_{train}$ and $\hat{S}_{test} = S_{test}$.

All of this has to be taken into account when applying both of those approaches in practice. We recommend that one could start with the online approach to yield results fast, while being aware of a potentially high false positive rate. The absolutely highest-scoring sequences can be treated with the highest confidence in the correctness of the classification results, while absolutely lower-scoring sequences should be held back for further analysis. In this step the visualization described in Section 4.6.2 can be of great help. After a certain number of test sequences has been collected, the models converge towards those of the batch approach and should be applied to the samples of S_{test} - at least to those held-back sequences. In practice the decision between the batch and the online approach further depends on the inter-sample similarity of the respective data, eventually requiring learning methods capable of online learning (e.g. [LGKM06]). Since our data has a high inter-sample similarity and thus a low expected model shift, we are using the batch approach, as it provides a sufficiently exact classification performance baseline, as well as easier to maintain and better interpretable classification models.

Creating both stateless and stateful feature space and projecting individual samples into it has a computational complexity linear to the number of tokens in S, $O(|\Phi(S,1)|)$. This computational complexity remains for the batch classification. The complexity becomes $O(|\Phi(S,1)| \cdot |S_{train}| \cdot |S_{test}|)$ in the case of $s_{train} = \hat{S}_{train}$ and $s_{test} = \hat{S}_{test}$, which means it still remains linear to the number of tokens - even when using the online classification algorithm with subsets consisting of single samples.

4.6.2 Interpretation of the Results

As the practical applicability of our method is highly relevant, we also focus on finding ways to increase the explainability and interpretability of the learning process and the results. Interpretability of the model and the samples in the defined feature space is very important to achieve a trustworthy model, as only such a model will be dependable enough to be applied in practice. We therefore include the capability to inspect the selected dimensions and weights of the model on whether they are reasonably chosen. We also describe a confidence ranking to estimate, which samples are more reliably classified correctly - and which are less reliable, and may thus need to be inspected closer. The research of the interpretation of models and results of learning methods is of increasing importance, as the adoption of complex models and methods of machine learning is becoming more widespread. Exemplary research is found e.g. in [RSG16, VKMG17], where the authors discuss methods for explaining the models and samples of various learning methods, and in [BBM$^+$15, BSH$^+$10, KSA$^+$17, MBKM13, MLB$^+$17, MSM18], where specifically interpretative methods and their evaluation for non-linear kernel methods and neural networks are discussed.

The use of linear kernels in our Support Vector Machines allows the interpretation of both the model and the classification results through analyzing the

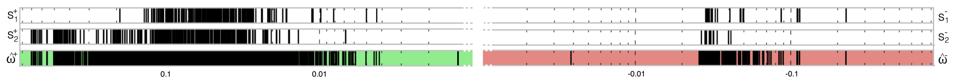


Figure 4.4: Positively contributing dimensions $\hat{w}^+$ (left) and negatively contributing dimensions $\hat{w}^-$ (right) of weight vector $\hat{w}$ and corresponding features of samples s_1 and s_2

weight vector w of the model, which is defined over the utilized feature space $\Gamma(S_{train})$, with $\Gamma \in \{\Gamma_{s.less}, \Gamma_{s.ful}, \Gamma_{comb}\}$. As the classification is done via the scalar product of w and s, the classification outcome is determined by either a few large weights, a large set of dimensions with smaller weights - or a combination of both. To find those dimensions, w can be analyzed directly, e.g. by sorting dimensions according to their weights in w, and then start with analyzing the dimensions with the largest values, assuming they are largely important. However, we are interested in those dimensions contributing most to the classification results - which are not necessarily those dimensions of w with the largest weights. Vector w of the trained model relies on the co-occurrence frequency of the dimensions found in S_{train}. Relevant dimensions which occur highly correlated within S_{train} will have lower weights in w, while relevant dimensions which occur uncorrelated in S_{train} will have higher weights in w. This influence can be removed by utilizing the covariance matrix over S_{train}, calculating $\hat{w}^T = w^T \cdot cov(S_{train})$, thereby increasing the interpretable weights of w for the highly correlated dimensions. This gives us a final new $\hat{w}$, which allows detecting absolutely relevant dimensions - irrespective of the underlying S_{train}. More details of this approach are discussed in [HMG+14]. By increasing the sparsity of vector w the interpretability can be further improved, as this removes the noise of dimensions with smaller weights. This can be achieved by using the L1 norm during model training, which reduces the overall number of support vectors and thereby the density of w, leaving only the most relevant dimensions for interpretation.

Figure 4.4 depicts an exemplary visualization of the interpretation of the weight vector $\hat{w}$. The black bars in the colored rows $\hat{w}^+$ and $\hat{w}^-$ depict the non-empty dimensions contained in the weight vector $\hat{w}$, which are contributing to a positive ($\hat{w}^+$) and a negative ($\hat{w}^-$) classification score, respectively. They are displayed on a log-scale, ordered descendingly from left to right. To increase the interpretability, the scales are centered around the bias b. The ranges of the scales are also limited, such that only the numerical range is covered which is also covered by concrete weights in w. For the examples s_1 of class -1 (invalid failures) and s_2 of class +1 (valid failures) this means, that dimensions matching with $\hat{w}^+$ (green area) increase the probability of the sample s to achieve a classification outcome $\langle w, s \rangle > b$, i.e. belong to valid failures (the positively labeled class). Dimensions matching with $\hat{w}^-$ (red area) increase the probability of the sample to achieve a classification outcome $\langle w, s \rangle < b$, i.e. belong to invalid failures (the negatively labeled class). With $s^- = \langle \hat{w}^-, s \rangle$ and $s^+ = \langle \hat{w}^+, s \rangle$ being the absolute sum of the negatively and the positively contributing matching dimensions, this corresponds to what we see for the exemplary samples s_1 and s_2, as $s_1^- > s_1^+$ indeed leads to a negative classification for s_1, whereas $s_2^- < s_2^+$ leads to a positive classification for s_2.

This visualization is specifically useful in practice, as already mentioned in Section 4.6.1. Once the test sequences are classified, the respective samples can be sorted according to their classification score. By plotting the corresponding $\hat{w}$ visualization for each sample at its side, we achieve an overview over all samples, with those dimensions highlighted that contributed most to the classification result. This allows to get a fast overview over the whole test set, which dimensions are shared between samples and which are most relevant for each of both classes. Due to its ordering it also helps in selecting potential candidates for a succeeding manual analysis, as those samples with the absolutely highest scores are already classified correctly with the highest probability, whereas those with a score close to the bias, i.e. samples close to the decision hyperplane of the SVM should either be inspected manually to assure a higher classification performance, or re-classified later with potentially better model trained on a larger set of S_{train}. Section 7.4.1 contains an additional discussion with more detailed examples illustrating this concept.

4.7 Evaluation

For the evaluation of the classification performances we are using the ROC-AUC values (i.e. the area under the receiver operation characteristics curve). Using the ROC-AUC allows us to calculate a performance metric over our unbalanced classes, because the ROC-curve relies on the percentage ratios of the *true* and *false* positive rates. As a result its AUC provides an upper bound on the classification performance, achievable by a model defined by choosing a specific point on the ROC-curve, corresponding to a specific compromise between false and true positive rates. Training a classifier for a certain value of true or false positive rate can be achieved by using a calibration process on a separate validation data set. Calibrating a classifier for a practical application always requires a preference decision: a low false positive rate or a high true positive rate. Since this decision strongly depends on the specific practical use case, which can prefer either one or the other, we are not conducting this calibration, but instead rely on the ROC-AUC value.

The method we introduced in Section 4.6.2 additionally allows interpreting the results of the classification process in practice, which is specifically relevant for the samples near the decision boundary and helps in further improving a selected model calibration process. While learning a calibrated classifier is easily done on our proposed classification methods by shifting the bias term b during the calibration, it is not easily possible on the competing learning methods we selected for the evaluation. Therefore, and because of its other advantages, we think directly using the ROC-AUC performance metric for performance comparisons is more suitable than using concrete values of false and true positive rates of calibrated models.

As described in Section 4.5, comparisons of different classification methods have already been conducted in other domains. Supervised learning using support vector machines often achieves good and robust results, which is why we designed our feature spaces to work with them. Additionally we also conducted experiments using unsupervised learning, specifically one-class support vector machines, to learn models representing valid failures, which can then be utilized to detect deviating samples, namely the invalid failures. However, the classification results proved to be worse than those of our supervised approaches, which is mainly caused by the circumstance that discrimination of both classes is much better when using features of both classes - and not only features of the dominating class. This high relevance of features specific to the class of invalid failures is also exemplarily visible in the visualization described in Section 4.6.2, where the average size of the weights of $\hat{\omega}^-$ is larger than those of $\hat{\omega}^+$.

The number of collected invalid failures is similar for both data sets (between 316 and 393 samples). The number of valid failures varies however, ranging from 4,045 to 4,550 for the 2014 data set, and from 11,743 to 11,887 for the 2015 data set. To adapt our evaluation setup to these circumstances and to assure comparability of the results, we randomly extract for each evaluation run the same number of samples for both data sets. We are using 300 samples of valid and 100 samples of invalid failures, split in equally sized training and test sets.

4.7.1 Evaluation Results

Section 4.3 explained how the test cases are designed to execute different processing chains on the underlying protocol layers and their inherent states and transitions, while Section 4.4 illustrates how the stateful feature space focuses on describing the transitions, while the stateless feature space focusses on describing details at the single states. Therefore we have a high variance in the properties of the 2014 and 2015 campaigns, huge differences in the their respective *debug*- and *tshark*-analysis data, and strongly varying size and expressiveness of the resulting *stateless* and *stateful* feature representations. As such we expect largely varying evaluation results over all those properties, which will be discussed now. The results of our evaluations on the 2014 campaign data are shown in Figure 4.5, whereas the results of our evaluations on the 2015 campaign data are shown in Figure 4.6. Additionally Table 4.4 shows the precise results achieved on both campaigns.

When analyzing the results, the Markov and Bayes classifiers show the lowest overall performance. In general the SVM and MLP classifiers on the stateful feature space perform slightly worse than those on the stateless feature space, which means the stateless features are describing the data better. Since the results using the stateless feature space outperformed the results on the stateful feature space, obviously its more detailed description of the individual states is more beneficial. While this underlines the importance of selecting proper features for any given data set, combining both types of features is even more beneficial and partly alleviates the need for such a decision. This is illustrated by the results on the combined fea-

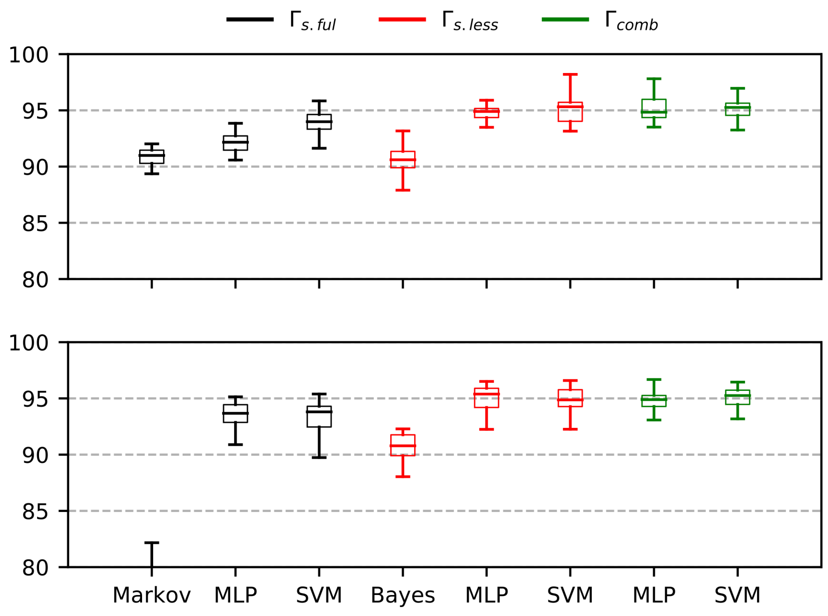


Figure 4.5: Box plots of the AUC-results (in %) of the comparative evaluation of the feature spaces and classifiers on the 2014 campaign data (top: *debug*-analysis data, bottom: *tshark*-analysis data).

ture vectors, which are consistently better than those on the individual feature vectors, having either the highest average results (2014 tshark with SVM, 2015 debug with SVM) or a good and practically relevant compromise of near-optimal average results and small variance (2014 debug with SVM, 2015 tshark with SVM).

When comparing the performances of Two-Class SVM and the MLP Classifier, a slight advantage for the SVM approach can be observed. Where on both campaigns both classifiers show a similar behavior in the results variance, the SVM largely outperforms the MLP in terms of average classification performance. While neural networks steadily increased their relevance and performance in recent years on varying types of data, obviously SVM classifiers are still a valid choice for this kind of data - with the additional advantages of easier interpretability and more reliable convergence. We can also see that the results on the *debug*-analysis data outperform those of the *tshark*-analysis data. While this was expected, given the more dynamic nature of the proprietary *debug*-analysis data format and its larger number of resulting features, the good performance that can be achieved with the more generic *tshark*-analysis data is surprising - especially as this is a freely available analysis tool.

When comparing the performances achieved on the difference campaigns, the results on the 2015 campaign data outperform those of the 2014 campaign data. Since we took care of utilizing identically sized sets of S_{train} and S_{test} for both

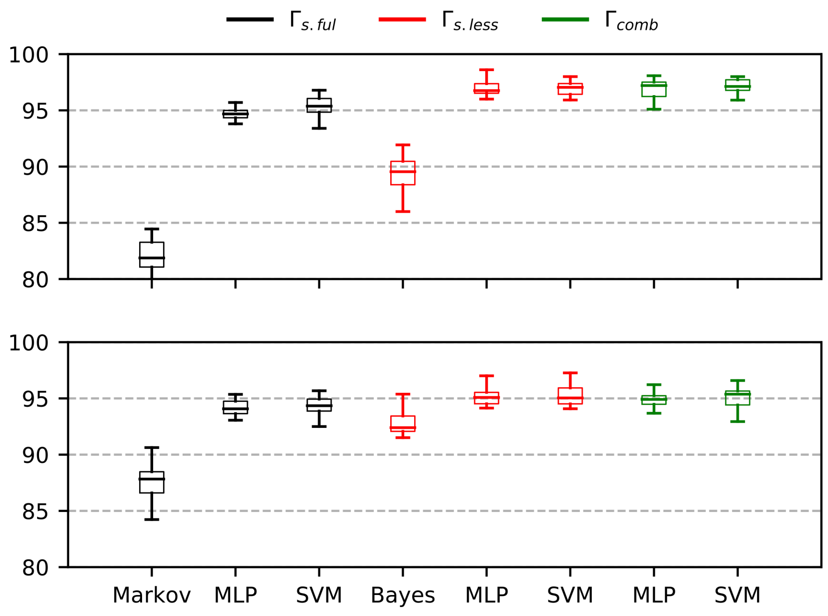


Figure 4.6: Box plots of the AUC-results (in %) of the comparative evaluation of the feature spaces and classifiers on the 2015 campaign data (top: *debug*-analysis data, bottom: *tshark*-analysis data).

campaigns, these differences originate in the different expressiveness of the two campaigns, represented by the differences in the number of tokens, as illustrated in Table 4.2 and Table 4.3. In the 2015 campaign better designed and more varying test cases lead to an improved classification performance on both types of analysis data, while the inclusion of additional keyword in the *debug*-analysis data further improved the expressiveness of the resulting feature vectors, and thus the final classification performances. Additionally the 2014 campaign contains more heterogenous types of invalid failures, making it harder to learn homogenous feature sets. In the 2015 campaign, invalid failures - which are caused by HTTPS traffic influencing the measurements - dominate. As they provide easier to learn feature characteristics, this also leads to improved results on the 2015 data. Furthermore we see differences in both feature spaces in that the debug features allow a higher classification performance on the 2015 data than on the 2014 data (with a difference of 1.9% with SVM on Γ_{comb}), while the corresponding performance difference on the tshark data is smaller (0.18% with SVM on Γ_{comb}). We see the main reason for this behavior again in the extended expressiveness of the debug 2015 features, which allow are more detailed description of the occurring behavior and hence more capable classification models. Similar to what we found in [SBKR12], we also see our expectation confirmed, that dynamic analysis - as used for the debug data - yields better classification results.

2014		Debug	Tshark
$\Gamma_{s.ful}$	Markov	90.96 ±1.30	76.83 ±4.46
	MLP	92.24 ±1.09	93.19 ±1.95
	SVM	94.07 ±1.26	93.34 ±1.51
$\Gamma_{s.less}$	Bayes	90.68 ±1.14	90.73 ±1.18
	MLP	94.75 ±$\underline{0.86}$	94.89 ±1.42
	SVM	$\underline{95.15}$ ±1.29	94.76 ±1.34
Γ_{comb}	MLP	94.99 ±1.29	94.75 ±$\underline{1.01}$
	SVM	$\underline{95.09}$ ±$\underline{1.08}$	$\underline{94.92}$ ±1.16

2015		Debug	Tshark
$\Gamma_{s.ful}$	Markov	81.88 ±1.56	87.70 ±1.59
	MLP	94.65 ±1.18	94.16 ±0.63
	SVM	95.28 ±1.14	94.32 ±1.18
$\Gamma_{s.less}$	Bayes	89.31 ±1.62	92.66 ±1.34
	MLP	96.82 ±0.88	$\underline{95.18}$ ±$\underline{0.81}$
	SVM	96.88 ±$\underline{0.72}$	$\underline{95.17}$ ±1.16
Γ_{comb}	MLP	96.91 ±0.88	94.70 ±1.10
	SVM	$\underline{96.99}$ ±0.84	$\underline{95.10}$ ±$\underline{0.98}$

Table 4.4: Mean AUC Results and Standard Deviation (in %) of the 2014 and 2015 campaigns.

4.7.2 Evaluation Interpretation

As discussed in Section 4.6.2, we are also interested in interpreting the results of the SVM-classification by analyzing the semantics of the most relevant dimensions of $\hat{w}$. Whereas the results discussed in the previous section are achieved with a token n-gram size of $n = 2$, using $n = 4$ lead to similar results, but allows a much better interpretation due to its larger context, which is why we are using the 4-grams for the analysis in this section.

In the models trained on the 2015 data set, the dominance of HTTPS connections leading to invalid failures is well represented in both feature spaces of debug and tshark data, with the respective features weighted highly negative within $\hat{w}$. In the models trained on the tshark data the most relevant dimensions in the stateful feature space are those state transitions that directly represent the use of HTTPS via the keyword TLS, like in `TLSv1 115` $\rightarrow$ `TCP 76` or `TLSv1 115` $\rightarrow$ `TLSv1.2 260`. In the models trained on the stateless feature space, the corresponding token 4-grams are similar, e.g. `TLSv1 2 260 Client` or `TLSv1 2 385 Application`. Besides those similarities between both feature spaces, there are also features that are unique to the respective feature space. In the stateful feature space different inter-protocol transitions are identified as causes for invalid failures, like `DNS 158` $\rightarrow$ `TCP 76` or `TCP 753` $\rightarrow$ `TCP 68`, whereas in the stateless feature token 4-grams of specific transmission sequences function similarly, e.g. `Seq 427 Ack 5` or `Seq 427 Ack 9`. In this way both feature spaces complement each other, as they focus on different aspects of the network behavior. In practice, an additional manual analysis could start with those specific ports or transmission sequences to find the cause and semantics of these invalid failures, or to verify the predictions of the classifiers, as such dimensions could also be an indicator for an overfitting during model training. Features contributing to a positive classification (i.e. valid failures) are less specifically defined, with state transitions and tokens like `HTTP/XML 996` $\rightarrow$ `TCP 1416` in the stateful feature space and `95 TCP 76 TCP` in the stateless feature space.

On the debug representation of the 2015 data we observe a similar behavior. There are dimensions that focus specifically on HTTPS and are contributing strongly to invalid classifications, like `Redirect to https //m` in the stateless feature space and

```
DBG Log HTTP_SP_EVAL GET → DBG Log HTTP_SP_EVAL Redirect
```

in the stateful feature space. Other very specific dimensions target concrete websites which reliably contribute to invalid failures, like the token 4-gram `youtube com service SMARTPHONE`, but most dimensions are less specific and would require a manual analysis to really identify their semantics. On the 2014 data the situation is similar, with the exception that invalid failures caused by HTTPS are not that dominant. As a result we see more general features, with the exception of quite specific token 4-grams in the debug representation which contain concrete error messages like `Session error SA failure` or `failure on TCP layer`. Obviously each feature spaces focusses on different highly relevant token n-grams, which - together with the overall better classification performance of the classifiers on the combined feature vectors - shows the potential of using those complementary feature spaces to achieve better overall classification performances and a better insight into the respective data semantics.

4.8 Practical Considerations

Besides the classification performance also the overall processing runtime can pose to be a *limitation* for practical applications. As discussed in Section 4.4, both feature spaces have a computational complexity linear to the number of tokens within the respective feature space. In terms of overall size and the amount of analysis required, the stateful feature space is much less complex than the stateless feature space, but still achieves a classification performance competitive to the stateless approach. In that regard it could be a viable option for systems which focus less on an optimal classification performance, but more on low computational complexity. For an optimal classification performance though, using the stateless or combined feature vectors is recommended. Selecting either batch or online processing in the feature extraction allows further adaptation of the computation to the requirements of any given practical application. Both methods are also easily parallelized due to their underlying vector operations. The same holds for learning and applying the proposed classification methods, rendering the complete processing chain a flexible and well performing solution for solving classification tasks on similarly structured temporal data.

4.9 Conclusion

In this chapter we have shown how to create stateless and stateful feature spaces from structural-temporal data, resulting in feature representations specifically designed to complement each other, to cover a wider spectrum of the properties defining the data, and to be suitable for a subsequent application of supervised learning methods. We discussed their properties and consequences for practical application and how to use them for discriminating data classes based on the different representation of their behavior. We competitively evaluated the classification performances on data representations of real-world data sets of mobile communication sequences, pre-processed using a non-proprietary and proprietary analysis methods. We showed that using the less expensive non-proprietary data analysis can enable feature representations nearly as expressive as proprietary ones, which widens the applicability of the proposed automatic learning approach. We also evaluated competitively the classification performance of various classifiers commonly used in the related fields, operating on the different feature space projections of the created data representations. A good classification performance was achieved both absolutely and in comparison to competitive approaches, specifically from the field of process mining, which highlights the suitability of the proposed analysis methods, features and learning methods for the otherwise manually solved problem of communication validation data classification. We also showed that the best final classification performance can be achieved by combining both stateless and stateful features. To further pronounce the practical applicability we also showed how to interpret the model to allow discriminating reliable from unreliable classification results, allowing to potentially decide for a succeeding manual analysis or later reclassification.

Besides their advantages, the proposed approaches also have different limiting requirements, like selecting appropriate values for the quantization and n to define the $\Gamma_{s.less}$ token n-grams, using data sufficiently expressive to provide state-defining token combinations for $\Gamma_{s.ful}$, adapting to the overfitting of the batch learning approach or the additional computational requirements of an online learning approach, or selecting the appropriate feature space based on the given computational restrictions. However, their advantages outweigh these limitations.

In terms of future work it could be promising to further enhance the complementary effects of both stateful and stateless features and create a better combined feature space, e.g. by gathering the most relevant stateless and stateful data properties directly during feature extraction. Further improvements could be thought of by integrating neural networks as learning methods. While interesting approaches like [MCCD13] allow learning on textual data using feedforward or recurrent neural networks, and methods like [ERF16, DL15] use LSTM to solve learning problems on process or sequence data, there are no approaches available which are capable of simultaneously handling the rich structural and sequential properties contained in the discussed data. These ideas will be analyzed in Chapter 5 for a slightly different data type.

Chapter 5

Feature Spaces and a Learning System for Structural-Temporal Data

Summary

The service quality and system dependability of real-time communication networks strongly depends on the analysis of monitored data, to identify concrete problems and their causes. Many of these can be described by either their structural or temporal properties, or a combination of both. As current research is short of approaches sufficiently addressing both properties simultaneously, we propose a new feature space specifically suited for this task, which we analyze for its theoretical properties and its practical relevance. We evaluate its classification performance when used on real-world data sets of structural-temporal mobile communication data, and compare it to the performance achieved of feature representations used in related work. For this purpose we propose a system which allows the automatic detection and prediction of classes of pre-defined sequence behavior, greatly reducing costs caused by the otherwise required manual analysis. With our proposed feature spaces this system achieves a precision of more than 93% at recall values of 100%, with an up to 6.7% higher effective recall than otherwise similarly performing alternatives, notably outperforming alternative deep learning, kernel learning and ensemble learning approaches of related work. Furthermore the supported system calibration allows separating reliable from unreliable predictions more effectively, which is highly relevant for any practical application.

5.1 Introduction

Sequences of structural and temporal data combine properties of symbolic sequences and multi-variate time series, in that a single sequence s of length r has the format $s = [(t(e_1), e_1), ..., (t(e_r), e_r)]$, i.e. each event e_i occurs at timestamp $t(e_i)$ within s, s.t. $t(e_i) = t(s)_i$. Additionally each sequence s can have a label $y(s)$. When trying to process such data with temporal properties (i.e. semantically relevant time intervals of varying length between individual events) and structural properties (i.e. a semantically relevant order and context of events and their represented behavior), research is often faced with different problems, like varying sequence lengths and the lack of feature spaces that allow representing both temporal and structural properties sufficiently well. With suitable feature spaces, processes that rely on such data can be better analyzed and represented, consequently allowing the application of a wide range of learning methods on those features.

This is true for all processes that create time-dependent structured data, like multi-layer protocol-based network communication, whose data can be recorded in real-time by logging systems. This allows the analysis of its structural-temporal data, utilizing its structural properties (e.g. protocol state-machine behavior) and its temporal properties (e.g. response timings) to continuously improve the quality of the respective communication service. The problem becomes even more complicated when analyzing the data of multi-directional real-time communication setups, as in video conferencing systems, in cloud infrastructures [SB14] or even in industrial infrastructure [KL14] networks. In those cases the temporal and structural properties are contained in multiple interacting event sequences, and the temporal properties are of vital relevance for the service quality of the system.

To be able to apply machine learning methods to solve the different types of problems occurring on such data (e.g. classification or prediction), specific feature spaces are required that properly represent those structural and temporal data properties and allow projecting sequences of arbitrary length onto feature vectors of homogenous length. To achieve this we analyze the structural and temporal properties of such highly time-dependent multi-client sequence data, and propose a new combined feature space of homogenous length. We also show with an extensive competitive evaluation and statistical analysis how this feature space succeeds in this task.

As practical use case for structural temporal data we focus on bi-directional multi-client real-time mobile communication data, used to solve the specific problem of detecting and predicting known sequence classes on data of both known and unknown sequence classes. On this data all of the previously mentioned properties are relevant, i.e. the order and context of the events are class specific and relevant, as are both the individual and the interacting sequences of both clients, as well as the temporal properties represented in the contained timestamps and their subsequences. For this use case we introduce and evaluate a system for the automatic classification of failures of communication sequences between mobile clients. This system allows supporting or replacing the expensive manual classification usually handled by domain experts. We conduct a detailed analysis of the practical implications and requirements, especially on how to calibrate the system for a high precision while achieving a reasonable effective recall. On this system we comparatively evaluate the classification performances when using the proposed feature spaces with baseline, as well as competitive deep learning, kernel learning and ensemble methods from the fields of process mining and sequence classification, allowing to draw implications on their general suitability and their individual advantages and disadvantages.

While we are focusing our analyses on the specific area of mobile communication, the proposed system for the detection and prediction of pre-defined classes of sequential data, as well as the proposed feature spaces can be applied in all areas working with structural temporal data, specifically for problems that require the incorporation of real-time or multi-client system properties. This enables more precise classifiers and reduces the amount of required manual analysis, whose expensive cost would otherwise prevent the scaling of such a classification system to large scale data sets.

Summarizing the primary objectives, we aim to overcome the restrictions of existing approaches of process mining and sequence learning by combining structural and temporal features, finally integrating all into one system. As such the contributions of this chapter are the following:

1. Comparison of the classification performance of features and learning methods commonly used in process mining and sequence classification, with the proposed combined feature space

2. Theoretical analysis of a baseline combined feature space against the proposed combined feature space

3. Proposal of a combined detection and prediction system for failure classification of structural temporal data of mobile communication, with an additional focus on the calibration of practically relevant metrics

Hence Chapter 5 is structured as follows: This introduction is followed by a discussion of the related work in Section 5.2. The use case and the utilized datasets are discussed in depth in Section 5.3, followed by the introduction of the relevant feature spaces in Section 5.4. Afterwards the components of the proposed system for sequence class detection and prediction are described in Section 5.5, including the utilized learning methods. This is followed by the evaluation and a statistical analysis of the classification system and the introduced feature spaces in Section 5.6, before finally reaching the conclusion in Section 5.7.

5.2 Related Work

We are interested in analyzing and evaluating features spaces representing the unique properties of sequences of structural, temporal data, whose inter-event *structural properties* have a semantically relevant relation to each other. We do this with a focus on mobile communication data, as an example of real-time protocol-based *network communication* data, where both the raw data logs and logs of additional dynamic analysis allow representing the contained structural dependencies. Further details on mobile communication protocol behavior can be found in [Sau10]. While similar analyses have been done for network communication, as e.g. in [CYLS07, CWKK09, CPWK06, CPC$^+$08, KGKR12, LJXZ08, NBFS06, WCKK08], most research focusses on the structural data properties, and less on the temporal properties relevant for the real-time execution of such processes.

This makes our research unique but also highly relevant for commercial processes, where such approaches are still highly sought after, e.g. in the form of *process mining* [VDAADM$^+$11]. In this field different objectives are solved on business analytics data, which often comes in a format similar to our, i.e. consisting of events and timestamps. As such [PNC11] are using Markov Classifiers to predict the time remaining to completion of a business case, which is also done by [PSBDL18] using Naive Bayes Classification. The verification of LTL (linear temporal logic) compliance is approached in [MDFDG14] by using Decision Trees. One large research topic is also the prediction of the next events during runtime, for which LSTMs [ERF16], Decision Trees [CLF$^+$14, LSD$^+$15, ULD16], Markov Classifiers [LGN12, LSD$^+$15, ULD16] or Multi Layer Perceptrons (MLP) [MLI$^+$15] are used. In the latest research projects, [TVLRD17] are using LSTM to simultaneously allow the runtime prediction of the next events and the prediction of the remaining time to case completion, [LWR$^+$10] are using MLP for detecting service level agreement violations, and [LCDF$^+$15] are using Markov Models and Decision Trees for predicting the achievement of a performance objective, or fulfillment of a compliance rule. While these approaches sound promising, they rely primarily on the structural information provided by the event sequence, and less on temporal data. The high dynamic interactivity between mobile network communication processes is also not easily represented by these approaches, as they require their training data to already contain all possible process behavior. Also these methods rely on large labeled training data sets, which can not necessarily be provided in our envisioned setup. Also none of the methods allows dynamically learning features relevant for predicting manually assigned class labels, which are per definition less clear defined than labels based on the violation of concrete event-driven constraints.

Since we are operating on sequential data, one could also use methods originally from the field of *sequence labeling*, where the task is to predict a label for each event within the sequence, which was solved using methods like Hidden Markov Models [Rab89], Conditional Random Fields [LMP01], MLP [Col02], but recently also Recurrent Neural Networks and specifically LSTM Neural Networks [DL15, Gra12, HGX$^+$17]. Since in our objective data each event is already labeled though, we are not interested in predicting such individual labels, but instead require predictions based on the behavior represented by the complete sequences. Using predictions of such individual labels to describe a complete sequence label is also no option, as a defined sequence label can depend on structural-temporal properties not represented in event-wise predictions. Since the event order and the inter-event durations are semantically relevant, using methods of sequence alignment [TLLP11] is also not directly possible. Also methods like dynamic time warping [KR05] are not directly applicable, as they are not designed to process temporal data with additional structural properties, or multi-client dependencies. As such we need methods producing a label for the whole sequence, as is done in *sequence classification* [XPK10, Die02]. Of those methods *support vector machines* [LSTCW01, LS05, SRS05a, SRS05b, SCW$^+$03, CV95, MMR$^+$01] are a well established method for *feature-based* sequence classification, which we are specifically interested in as we are trying to reflect the data properties by specific feature spaces. SVM approaches have shown a reliable performance, specifically when using non-sequential n-gram feature spaces for the respective sequential data, as done in the fields of natural language processing [BPX$^+$07, GSZ13, PSC15, WM12], intrusion detection [RTWH11, Rie09], malware detection [RKD10, LŠ11, SBKR12] and the analysis of network communication [ORLS14, PAF$^+$09, WPS06, WS04].

For our feature spaces we are using different types of features to represent the temporal and structural data properties within the feature spaces. Specifically we are relying on token n-grams, i.e. sequences of n arbitrary tokens. This feature representation originates in the field of natural language processing [BPX$^+$07, GSZ13, PSC15, WM12], but has also been extended to network communication [RTWH11, Rie09, ORLS14, PAF$^+$09, WS04]. However, we are extending this structural feature type by including additional temporal information, and by also integrating a wider context for each token n-gram, an idea similar to the integration of additional context information as introduced in [MCCD13] for the CBOW (continuous bag of words) and Skip-gram models.

All of this shows that our approach can be clearly distinguished from other approaches of related work. Additionally our proposed system combines supervised methods for detecting and predicting problem classes in a novel way, allowing a calibration to the metrics relevant in this specific problem domain, similar to what has been done in [MLI$^+$15].

5.3 Use Case Description

We will now discuss the properties of the data and the contained problem classes of our concrete use case of failure classification for sequence data of mobile communication, and why these are representative for the analysis of structural and temporal aspects.

5.3.1 Data Set Properties

We are using mobile communication data of a specific format as a concrete example to discuss and show the properties of sequential structural and temporal data. It was recorded to provide a wide-ranging quality analysis of the underlying network infrastructure. The data is collected by a fleet of specialized cars equipped with roof-mounted antennas and multiple android smartphones. The mobile phones run test sequences, which consist of automatically calling a phone within one of the other cars, establishing a connection, playing a voice chat of 1 minute and finally closing the connection. This whole process is monitored, recorded and consecutively analyzed by a system which focuses on various key performance indicators (KPIs) and statistics, radio frequency (RF) values and the successful completion of the key processing steps of the respective protocol state machines of UMTS, LTE and GSM. Since the cars are moving while all of this takes place, the recorded data also contains switches between different transmission technologies (e.g. the sequence may start in GSM, switches then to LTE, and finishes in UMTS). The resulting data already allows for drawing simple conclusions, e.g. on whether communication sequences have been successful at all (i.e. the relevant KPI and RF values did not show negative deviations, and all relevant protocol states have been completed successfully), whether they dropped in the middle (i.e. important protocol states have not been completed), or whether they have failed for other reasons.

Those failed sequences are then manually analyzed to determine their reason of failure and potentially even its cause. This process is called *failure classification*, allowing to assign a specific failure class to a failed sequence. Doing this manually by looking at many hundred log file entries is a tedious, time-consuming and expensive task. By using statistics over the KPIs, RF values and the protocol states for rule-based approaches, this can partially be automated, specifically for failures with simpler behavioral patterns. A more versatile machine learning approach could however help solving this problem better, potentially allowing to cover less clearly defined failure classes, while also adding some flexibility when trying to find new failure classes, caused by to changes in the communication technology backbone architecture, e.g. with the upcoming 5G [CFH14] technology.

For our analyses we collected two different data sets: the MFC data set, containing manually labeled and unlabeled failure class samples, and the AFC data set, containing failure class samples which are automatically labeled by a rule-based approach, as well as unlabeled samples. Table 5.1 shows some of their most important characteristics. Each contained sample represents a call sequence, which utilizes at least the GSM, UMTS or LTE protocol, following its respective specification and call phases, which is reflected in the logged events and the respective timestamps. Instead of using all the recorded events though, we are mainly interested in those that are relevant for the potential failure classes. Hence we are using filters based on rules that have been defined by experts with deep domain knowledge, removing all events that contain redundant or irrelevant information. As a result we obtain a final set of base events, which together with their different states (e.g. different reasons for a location update reject), and in combination with the respectively utilized protocol, leads to overall 335 different event types.

Both data sets will serve different purposes in Chapter 5, because the details of the different labeling approaches used for both data sets are expected to impact the AFC evaluation performance negatively. The filtered events available in both data

	MFC		AFC	
Average $e_{\#}$	44.11 ± 13.51		29.75 ± 27.24	
Main failure classes	$c_{\#}$	$s_{\#}$	$c_{\#}$	$s_{\#}$
CSFB Failure	2	48	2	370
Congestion Failure	-	-	1	60
Core Network Failure	1	30	8	682
E2E Failure	1	19	2	169
UE Failure	1	21	-	-
Other Failures	39	86	39	360
Sum	44	204	52	1,641
Unlabeled Samples	-	5,873	-	1,623

Table 5.1: Data set statistics: Number of events $e_{\#}$, of failure sub classes $c_{\#}$ and of samples $s_{\#}$

sets are selected to cover all important call phases and event sequences potentially relevant for a proper class prediction. As such they are theoretically sufficient to properly reproduce the MFC labels. However, they are insufficient to achieve a similar performance for the AFC data, where also data of additional events, as well as relevant KPI and RF data has been used for defining the classes, none of which we have access to in our structural-temporal data. As a result of these restrictions Table 5.1 shows that the average number of events per sequence is much smaller for AFC data, and its variance is much larger, reflecting a larger class variance and making a proper discrimination harder. Additionally, the AFC data contains 8 very similar variations of the *Core Network Failure* class, which are harder to discriminate as well. Due to these shortcomings of the AFC data set, the smaller MFC data set is more relevant for our purpose, as its labels provide a better ground truth. Since this is a problem in the AFC data set, we will restrict our analyses on the AFC data to those which are expected to provide valuable insights on this data set only, namely how well the proposed feature spaces and learning methods can still reproduce the AFC labels under these conditions, and - given its larger number of samples - how much of a performance improvement we can expect when increasing the training data size.

We will now discuss the format of the contained sequences. Each sample in our data sets consists of the bi-directional communication sequence between a caller and a callee. We denote the caller as the MOC client (mobile originated call) and the callee as the MTC client (mobile terminated call). The samples contain highly structured sequential data (order of events) with highly relevant temporal components (inter-event durations), all of which are semantically relevant for discriminating the failure classes, e.g. reflecting call phases being incomplete or too long, or reflecting an anomalous order of events. Table 5.2 shows an exemplary event sequence of a successful call, starting in LTE and proceeding and ending in

UMTS. This example also introduces the new timestamp format $t_{s.rel}$, denoting sequence-relative timestamps, defined for a sequence s as $t_{s.rel}(e_i) = t(e_i) - t(e_0)$ (or $t_{s.rel}(s)_i = t(s)_i - t(s)_0$), i.e. the timestamp of the first sequence event is set to 0.0, and all other timestamps are reset relative to this value. In the example the MOC client is set up until $t_{s.rel} = 12.232$, then the MTC client is set up until $t_{s.rel} = 14.231$. Once the clients are connected at $t_{s.rel} = 30.103$ the call takes place, before they are finally disconnected again. The abbreviations represent the following event types: extended service (ES) request, security mode (SM) command and complete, connection management service (CMS) request, radio bearer (RB) setup and extended service (ES) request.

5.3.2 Failure Classes

Before analyzing how the concrete sequence properties are utilized in the feature spaces, we first need to discuss the existing failure classes and their defining structural and temporal properties in more detail, to provide a better practical background of the problem domain. Table 5.1 contains an overview and provides relevant class statistics for the selected, sufficiently sized failure sub-classes of the listed main failure classes. It also contains details about the samples of insufficiently sized failure classes, grouped together in the set of *Other Failures*, as well as the additional number of unlabeled samples per data set.

CSFB Problems

A circuit switch fallback is conducted when the current network (e.g. LTE) does not sufficiently fulfill the current connection requirements (e.g. signal strength, cell coverage, sufficient response times) or suffers from other problems, while at the same time an older network (e.g. GSM) is available. It can also occur when one of the communicating clients is not LTE capable. Handing over the correct connection state to another protocol can be problematic, though. Our data set contains cases of failures that occurred when the call setup was not properly continued after the location area update (LAU) and the routing area update (RAU), when the current network did not allow a proper release for redirection, or when the redirection to the older network simply took too long.

Congestion Problems

These problems can occur when the network is overloaded, s.t. problems with the connection or response timings occur. In our data sets we have sample sequences where the connection downlink disconnected too early, leading to an interrupted connection, and other cases, where no circuit channel was available, completely preventing to establish a connection.

	MOC client		MTC client	
$t_{s.rel}$	protocol	event	event	protocol
0.0	LTE	ES Request		
1.271	LTE	SM Command		
1.271	LTE	SM Complete		
2.385	UMTS	SM Command		
2.386	UMTS	SM Complete		
3.298	UMTS	CMS Request		
3.864	UMTS	SM Command		
3.864	UMTS	SM Complete		
4.273	UMTS	Setup		
4.826	UMTS	Call Proceeding		
5.192	UMTS	RB Setup		
5.392			ES Request	LTE
6.209			SM Command	LTE
6.209			SM Complete	LTE
12.232	UMTS	RB Setup Complete		
	MOC client is set up			
12.577			SM Command	UMTS
12.577			SM Complete	UMTS
13.130			Setup	UMTS
13.253			Call Confirmed	UMTS
13.783			RB Setup	UMTS
14.231			RB Setup Complete	UMTS
			MTC client is set up	
14.799			Alerting	UMTS
14.865	UMTS	Alerting		
15.721			Connect	UMTS
16.254			Connect Ack	UMTS
18.237	UMTS	Connect		
19.020	UMTS	Connect Ack		
29.923	UMTS	SM Command		
29.924	UMTS	SM Complete		
30.032	UMTS	RB Setup		
30.103	UMTS	RB Setup Complete		
	Clients are connected and the call takes place			
93.028	UMTS	Disconnect		
93.427			Disconnect	UMTS

Table 5.2: Exemplary sequence of successful bi-directional mobile communication

Core Network Problems

This failure class represents more general problems, where the causes might be similar to those of the previous failure classes, e.g. an unexpected downlink disconnect or problems with LAU and RAU. The previous classes, however, contain additional semantic properties that lead to the classification as CSFB (fallback to older technology) or congestion problem (high latency, low bandwidth), which are missing here. Failure sub classes dominant in our data sets contain cases of unexpected downlink disconnect, unreachable MTC clients, or no or too slow reply to LAU or RAU. The high similarity to other classes, as well as the fact that only minor differences exist between its individual sub classes makes discriminating them harder, which is specifically relevant for the AFC data, with its higher number of samples of these sub-classes.

E2E Problems

End to end problems occur beyond the scope of the core network. In our data set two of its sub classes are prominently represented. Unexpected downlink (DL) radio resource control (RRC) connection releases are symptoms of problems during the downlink authentication phase of the connection, causing it to fail. This also holds for missing downlink setup failures, which already fail at an even earlier state.

UE Problems

Besides those network and protocol related failures, problems can also occur on the devices themselves. The MFC data set contains samples of potential firmware issues, leading to such problems.

Other Failures

Sequences of failure classes that contain only very few samples are not useable for a proper evaluation. Those samples are all re-labeled as *Other Failures*, allowing to use them in the model class detection step of our system.

5.4 Structural and Temporal Feature Spaces

One of the objectives of Chapter 5 is to analyze feature spaces capable of representing sequential data with structural and temporal properties, like the one detailed in the previous section, and to propose a feature space suited to better represent those properties. To achieve this, we discuss five different feature spaces Γ_{qT}, Γ_T, Γ_S, Γ_{S+T} and Γ_{ST}. Γ_{qT} is based on a sequential representation of the data, as commonly used in process mining [VDAADM$^+$11]. Since we are specifically interested to additionally integrated temporal information in our feature spaces, Γ_{qT} optionally allows the inclusion of quantized temporal information. Γ_T focusses on the temporal information in a re-ordered sequential representation, while Γ_S focusses on the structural information in a non-sequential representation, essentially using a token n-gram approach, as described in Section 5.2. Finally we propose Γ_{S+T} and Γ_{ST} to show the advantages of integrating structural and temporal information complementarily into a single feature space, which is expected to allow a better data representation, compared to using only structural or temporal features alone. We discuss those feature spaces abstract, but also discuss unique properties that are specifically relevant for our use case. As such some of those discussions are exemplary adaptations of the abstract feature space properties to our concrete use case. However, this should not be seen as a restriction on the general applicability of these feature spaces, as they can be adapted to other use cases as well.

Nomenclature

MC_i	Model class i, i.e. sequence class sufficiently sized to train a classifier
$s_{\#MC}$	Configuration parameter for the number of samples in this model class
$\|s\|$	Number of events in a sequence s
$\neg MC$	non Model Class, containing samples of insufficiently sized classes
MCP	Model Class Predictor
MCD	Model Class Detector
Γ_{qT}	Quantized temporal feature space
Γ_T	Temporal feature space
Γ_S	Structural feature space
Γ_{S+T}	Concatenated structural temporal feature space
Γ_{ST}	Structural temporal feature space
S	Set of sequences s
S^M	Set of model sequences s^M, defining the Γ_{ST} feature space model
Θ_{MCD}	Confidence rating based on the MCD
Θ_{db}	Confidence rating based on the decision boundaries of the MCP
θ_{db}	Calibration parameter for the MCP decision boundaries

5.4.1 Base Processing

All of the following feature spaces require an identical base processing, for which we need a second timestamp format, enabling the consideration of relative time-dependencies (i.e. local delays): the event-relative timestamps $t_{e.rel}$, defined as $t_{e.rel}(e_i) = t_{s.rel}(e_i) - t_{s.rel}(e_{i-1})$ (or $t_{e.rel}(s)_i = t_{s.rel}(s)_i - t_{s.rel}(s)_{i-1}$) for a given sequence s.

One objective for our proposed feature spaces is, that they represent multi-client behavior within the sequential data. This is relevant for our use case, because some failures can be caused by erroneous behavior on the MOC side of the call, while others are caused by problems on the MTC side - or even by problems on both sides, individually or combined. To reflect those structural properties, we need to create the sub-sequences s_{MOC} and s_{MTC} to contain exclusively the events of MOC and MTC respectively, with $|s_{2C}| = |s_{MOC}| + |s_{MTC}|$. These representations allow the feature spaces to omit events of the respective opposite client. Table 5.2 shows this behavior exemplarily for the MOC-events at $t_{s.rel} = 5.192$ and $t_{s.rel} = 12.232$. They are interrupted by three MTC events in the s_{2C} sequence, but are represented consecutively in the s_{MOC} sequence. As a result, a sequence s can be described by a triple of sequences s_{2C}, s_{MOC} and s_{MTC}. If not stated otherwise, we use s synonymously with s_{2C} to denote the complete 2-client communication. As such the example in Table 5.2 effectively illustrates an s_{2C} sequence. This definition is extended to the whole set of sequences. Where previously we denoted all sequences s in a set S as $s \in S$, we can now also denote the sets of sequences of each different representation, i.e. $s_{2C} \in S_{2C}$, $s_{MOC} \in S_{MOC}$ and $s_{MTC} \in S_{MTC}$. This also allows extending the definition of $t_{s.rel}$ to those representations, in that $t_{s.rel}(s_{2C})$ denotes the vector of the sequence-relative timestamps of s_{2C}, and $t_{s.rel}(s_{MOC})$ and $t_{s.rel}(s_{MTC})$ denote those of the client-wise sequence representations. The same holds for $t_{e.rel}$. Note that $t_{s.rel}$ is set to 0.0 for the first event of each sequence representation respectively (i.e. $t_{s.rel}(s)_i = t(s)_i - t(s)_0$ for $s \in \{s_{MOC}, s_{MTC}\}$), to allow for a better comparability of sequences of the same client type.

Using an exemplary set of event identifiers $E = \{'A', 'B', 'C', 'D'\}$ allows creating two artificial example sequences x_1 and x_2 and their timestamps in the two formats, as shown in Table 5.3. These will be used in the next sections to illustrate various aspects of the different feature spaces.

x_1			x_2		
$t_{s.rel}$	$t_{e.rel}$	e_i	$t_{s.rel}$	$t_{e.rel}$	e_i
0.0	0.0	D	0.0	0.0	D
2.211	2.211	A	1.823	1.823	B
4.341	2.130	B	3.230	1.407	B
6.207	1.866	C	5.103	1.873	C
18.075	11.868	A	25.330	20.227	B
			30.548	5.218	A

Table 5.3: Two artificial communication sequences x_1 and x_2

5.4.2 Γ_{qT} Features

Γ_{qT} features are based on the $s \in S_{2C}$ feature vectors, essentially representing the most common type of data representation used in the related work of process mining. We extend this representation additionally by $\underline{q}$uantized $\underline{t}$emporal features. The idea is to get a sequential representation of the event identifiers, which is specifically suited for classifiers used in process mining, but to additionally include temporal information. To achieve this, we start with the event indices in

s. Consecutive events with a temporal distance closer than a predefined minimum interval θ_{mi} maintain their current position in the event index sequence. If consecutive events exceed θ_{mi}, an additional empty event $e_{\emptyset}$ is inserted, reflecting the larger interval between those consecutive events. This is repeated until the next event is reached. As a result, smaller values of θ_{mi} introduce more events $e_{\emptyset}$ and lead to larger, sparser feature vectors, while larger values of θ_{mi} introduce less empty events, thereby decreasing the feature vector length while increasing the density - until a completely dense feature vector is achieved, containing no empty events $e_{\emptyset}$ at all. Via model selection a value of $\theta_{mi} = 5.0s$ is selected as a compromise capable of filling large temporal gaps occurring in data (which often represent overly long durations between two protocol states) while not increasing the overall feature vector length too much in less relevant regions of the sequence. For $\theta_{mi} = 5.0s$, the resulting Γ_{qT} feature vector for sequence x_1 in Table 5.3 is thus $\Gamma_{qT}(x_1) = ['D',' A',' B',' C', e_{\emptyset}, e_{\emptyset},' A']$, s.t. the large $t_{e.rel}('A')$ is represented by two instances of $e_{\emptyset}$. Choosing $\theta_{mi} > 60.0s$ allows eliminating all occurrences of $e_{\emptyset}$, which is identical to the original sequence of events, without the additional temporal information provided by the $e_{\emptyset}$ inserted in the sequence. When using Γ_{qT} in this way, we denote it as Γ_{qT^*}, which allows highlighting the performance differences when using both types of feature representations with competing classifiers of the process mining domain.

5.4.3 Γ_T Features

To create the set of *t*emporal features Γ_T, we use the set S_{2C}. The idea of Γ_T is to create a feature space which projects properties of the ith occurrence of each event type in a sequence s onto the same dimension. The projected property is the timestamp $t_{s.rel}$ of the respective event, allowing to compare it with the timestamps of the ith occurrence of the same event type of other sequences. We start by calculating the occurrence frequency $f(e, s)$ for each event type $e \in E$ in each sequence $s \in S_{2C}$. Then we define its maximum value as $m_e = max(f(e, s)), \forall e \in E, \forall s \in S_{2C}$, calculating what is the most frequent occurrence of event type e in any sequence. Furthermore a function $\kappa(e, s, i)$ is required, returning the ith occurrence of e in s. For example the simple sequence x_1 in Table 5.3 has two occurrences of the event type 'A'. As such, $\kappa('A', x_1, 2)$ returns $e_5 =' A'$, i.e. the 5th event in s is the 2nd occurrence of 'A' in s, with $t_{s.rel}(\kappa('A', x_1, 2)) = 18.075$. Now a function $ts(s, e, m_e)$ can be defined, providing a vector of these timestamps $t_{s.rel}$ of all occurrences of a selected event in a sequence, and 0.0 otherwise:

$$ts(s, e, m_e) = [\tau(s, e, 1), ..., \tau(s, e, m_e)]$$

with

$$\tau(s, e, i) = \begin{cases} t_{s.rel}(\kappa(e, s, i)) & \text{if } i \leq f(e, s) \\ 0.0 & \text{else} \end{cases}$$

Concatenating the resulting vectors of $ts(s, e, m_e)$ for a sorted list of all $e \in E$ via the concatenate()-function yields the final feature vector for a sequence s, which has for all samples the same length of $\sum m_e, \forall e \in E$. A small example should make this better understandable. Table 5.3 shows two simple sequences $\{x_1, x_2\} = X$, for which the maximum frequencies $m_e = max(f(e,x)), \forall x \in X$ of each $e \in \{'A','B','C','D'\}$ are $m_A = 2$, $m_B = 3$, $m_C = 1$, $m_D = 1$. As a result the feature vectors have the following format:

$$\Gamma_T(s) = \text{concatenate}([ts(s,'A',2), ts(s,'B',2), ts(s,'C',1), ts(s,'D',1)]),$$

for $s \in \{x_1, x_2\}$, resulting in the following final feature vectors:

$$\Gamma_T(x_1) = [2.211, 18.075, 4.341, 0.0, 0.0, 6.207, 0.0]$$

and

$$\Gamma_T(x_2) = [30.548, 0.0, 1.823, 3.230, 25.330, 5.103, 0.0].$$

5.4.4 Γ_S Features

The structural Γ_S features are event n-gram features, similar to the previously used token n-gram features, and as such representing a feature representation commonly used in the related works of sequence classification. Therefore we extract for all s_{2C}, s_{MOC} and s_{MTC} all event n-grams and index their sorted list, spanning the final feature space Γ_S - including the client-specific n-grams of s_{MOC} and s_{MTC}. We denote an event n-gram of a sequence feature vector s via its vector indices in interval notation, i.e. $s_{[i,i+n)}$ denotes the event n-gram from position i (inclusive) to position $i+n$ (exclusive). The Γ_S feature vector of sequence sample s is then defined via the binary occurrence of the respective event n-gram within s. When using the examples x_1 and x_2 from Table 5.3, the sorted list of n-grams for $n = 3$ is ['DAB', 'ABC', 'BCA', 'DBB', 'BBC', 'BCB', 'CBA'], resulting in the final Γ_S feature vector $\Gamma_S(x_1) = [1,1,1,0,0,0,0]$.

5.4.5 Γ_{S+T} Features

One base hypothesis of Chapter 5 is that the classification performance can be increased by using a complementary structural and temporal feature space. For the structural-temporal Γ_{S+T} feature space we treat the feature vectors of Γ_S and Γ_T as equivalent. Because of it binary format, Γ_S already produces qualitative feature vectors, but Γ_T produces quantitative feature vectors. If we binarize its values, we create a qualitative representation, which we can simply concatenate with the Γ_S feature vector. This is used here to provide a baseline complementary feature space, before defining the more complex complementary feature space Γ_{ST}.

5.4.6 Γ_{ST} Features

For the structural-temporal Γ_{ST} feature space we will first need an analysis of the representative capabilities we specifically want to achieve with this feature space. As such, we will start this section with an analysis of some feature requirements, before explaining how these requirements are met by creating the data representation via structural-temporal $\delta - n$ matching and the use of model sequences.

Context and Position

Metrics and features for structured, sequential data should reflect its specific properties. A sample of such data could be described by the occurrence of single n-grams (as done in Γ_S). But this description can be improved when these n-grams are also analyzed in terms of their broader context and position. As such two similarly positioned n-grams might be identical, but their respective neighbor events (their context) might be different, which should prevent or penalize a match between them. This is highly relevant in data which is created by protocol-driven processes, like mobile communication data, which follows specific protocol states (e.g. for the radio bearer setup or the security parameter negotiations), all requiring specific events in their context. Thus it is important to focus on comparing contextualized n-grams with each other, i.e. events at the call setup should not be compared with those in the final call phases.

Model Sequences

For the definition of Γ_{ST} the concept of *model sequences* needs to be introduced. Projecting each of the $s \in S$ onto a feature space spanned by these model sequences yields projected samples of the same length, while at the same time incorporating both temporal and structural properties. The use of model sequences is based on the idea of defining the features of a sequence s based on its similarity to each model sequence s^M in the set of model sequences S^M, which thus defines a feature space *model*. To this purpose we define the set of model sequence representations as a triple $S^M = (S^M_{2C}, S^M_{MOC}, S^M_{MTC})$ just as we did for our actual sequences. By consecutively indexing the sequences and the events within S^M we effectively span a feature space of size $\sum_{\forall s^M \in S^M} |s^M|$. Note that the model sequences do not have to be labeled, and also do not have be mutually exclusive to the set of training or test sequences $S = \{S_{2C}, S_{MOC}, S_{MTC}\}$, as we are not using the labels of the model sequences in any way. We rely instead on the relevance of their contained structural and temporal properties, offering insight into relevant types of behavior, required for the class discrimination. However, as the feature space is spanned by using the model sequences, their labels could potentially be used to increase the contained number of different features, or to balance the representation of features of more complex failure classes against those of simpler ones.

Defining the Structural Temporal Features

The Γ_{ST} feature space is based on the idea of representing structural and temporal properties of the respective sequences. In this paragraph we will discuss, how to achieve this by using n-grams and model sequences in structural-temporal matching procedure, with a focus on the feature properties of context and position. We define the context of each event by the size of the n-grams and the parameter of positional variance δ, and the positional properties by the actual matching procedure. The idea of this procedure is to match each n-gram of each $s \in S$ with the n-grams of each $s^M \in \text{sorted}(S^M)$, requiring identical positions of both matching n-grams within the two sequences - and then loosen this requirement via the parameter δ. The result of this matching for each $s \in S$ is a vector of its structural similarity to each of the model sequences s^M, for each of its sequence representations s_{2C}, s_{MOC} and s_{MTC}. Specifically the additional matches on the S^M_{MOC} and S^M_{MTC} are relevant for failures, as they allow detecting event chains of a single client, automatically crossing the gap caused by interfering events of the other client. We achieve this by a structural δ-n matching function, denoted as $\hat{\Phi}(s, s^M, \hat{s}^M, \delta, n)$, where $\hat{s}^M$ denotes a zero-vector of length $|s^M|$. By iterating over all indices, this vector is populated as follows:

$$\hat{\Phi}(s, s^M, \hat{s}^M, \delta, n) = \begin{cases} \text{inc}(\hat{s}^M_{[i+j,i+n+j)}, \vec{1}) & \text{if } s^M_{[i+j,i+n+j)} = s_{[i,i+n)} \\ \hat{s}^M_{[i+j,i+n+j)} & \text{else} \end{cases}$$

for $\delta \geq 0$ with $i = [1, |s^M| - n + 1]$, $j \in [-\delta, \delta]$, $\forall(i + j) \geq 1$ and $\forall(i + n + j) \leq |s^M| + 1$, with $\text{inc}(\vec{x}, \vec{y}) = \vec{x} + \vec{y}$ denoting here the element-wise incrementation of vector $\hat{s}^M$ by a one-vector $\vec{1} = (1, ..., 1)$ of length n. Here we are using the indexing method introduced for Γ_S to denote individual n-grams, s.t. $s_{[i,i+n)}$ denotes the n-gram of the events $[e_i, ..., e_{i+n-1}]$ in the sequence s. As such we essentially compare $s_{[i,i+n)}$ with $s^M_{[i+j,i+n+j)}$, and allow a positional variance in the model sequence by defining j over the range of $[-\delta, \delta]$.

Using the exemplary sequence x_1 and x_2 of Table 5.3, with x_1 as

$$s = ['D', 'A', 'B', 'C', 'A']$$

and x_2 as model sequence

$$s^M = ['D', 'B', 'B', 'C', 'B', 'A'],$$

the structural δ-n matching with $\delta = 1$ and $n = 1$ yields the vector

$$\hat{\Phi}(s, s^M, \hat{s}^M, 1, 1) = [1, 1, 1, 1, 0, 1],$$

with event 'B' and the final 'A' making use of the positional variance introduced by δ.

Since we are not only interested in the structural properties of our data, we will now extend $\hat{\Phi}$ by integrating the event-relative timestamps $t_{e.rel}$ as temporal properties, obtaining the final structural-temporal projection function Φ. The idea is to calculate the absolute differences of the $t_{e.rel}$ of the structurally matching events of s and s^M. This is done by modifying the previously used incrementation function int(), giving rise to the final definition of Φ:

$$\Phi(s, s^M, \hat{s}^M, \delta, n) = \begin{cases} \text{inc}(\hat{s}^M_{[i+j,i+n+j)}, \Delta_{abs}(\vec{x}, \vec{y})) & \text{if } s^M_{[i+j,i+n+j)} = s_{[i,i+n)} \\ \text{ with } \vec{x} = t_{e.rel}(s^M)_{[i+j,i+n+j)} & \\ \text{ and } \vec{y} = t_{e.rel}(s)_{[i,i+n)} & \\ \hat{s}^M_{[i+j,i+n+j)} & \text{else} \end{cases}$$

for $\delta \geq 0$ with $i = [1, |s^M| - n + 1]$, $j \in [-\delta, \delta]$, $\forall(i + j) \geq 1$ and $\forall(i + j + n) \leq |s^M| + 1$, with the function $\Delta_{abs}(\vec{x}, \vec{y})$ defining a vector by calculating the absolute element-wise difference between two vectors $\vec{x}$ and $\vec{y}$, which is in this case the temporal difference of the respective matching events of s and s^M. As this can result in multiple matches per event in $\hat{s}^M$, we finally average each field in $\hat{s}^M$ by its number of matches. Applying this final formulation to the previous example sequences and their event-relative timestamps

$$t_{e.rel}(s) = [0.0, 2.211, 2.130, 1.866, 11.868]$$

and

$$t_{e.rel}(s^M) = [0.0, 1.823, 1.407, 1.873, 20.227, 5.218]$$

yields the final feature vector

$$\Phi(s, s^M, \hat{s}^M, 1, 1) = [0.0, 0.307, 0.723, 0.007, 0, 6.65].$$

Since Φ is defined over a single s^M, it has to be executed for all s^M, with $s^M \in \text{sorted}(S^M)$, and the resulting vectors $\hat{s}^M$ have to be concatenated via the concatenate()-function, giving rise to the final definition of Γ_{ST}:

$$\Gamma_{ST}(s, S^M, \delta, n) = \text{concatenate}(\Phi(s, s^M, \hat{s}^M, \delta, n)), \forall s^M \in \text{sorted}(S^M)$$

The final feature vector $\Gamma_{ST}(s, S^M, \delta, n)$ has the length $\sum_{\forall s^M \in S^M} |s^M|$, and contains the *structural-temporal* $\delta - n$ *matches* of the sequence s and all model sequences S^M. The O-complexity of this whole process is linear, as creating the feature space by indexing the model sequences S^M is done in linear time, and projecting a single sequence s onto S_M is linear to the number of model sequences $|S_M|$, as it requires matching the n-grams of each s with those of every $s^M \in S^M$.

Explaining the Semantics

The objective of our projection Φ is to achieve features highlighting differences in structurally similar, but temporally different sequences, i.e. we aim for a way to define similar features for sequences with similar structural and temporal behavior, while achieving different feature vectors for those which are structurally different, or which are structurally similar, but temporally different. As one can see, the value of a single dimension is 0 if there is no structural match, it is very small if

$t_{e.rel}(s_{[i,i+n)})$ and $t_{e.rel}(s^M_{[i+j,i+n+j)})$ are similar, and it is large if $t_{e.rel}(s_{[i,i+n)})$ and $t_{e.rel}(s^M_{[i+j,i+n+j)})$ strongly deviate.

Once the feature vector of s is calculated, it is utilized in the classifier, where this feature projection does indeed allow focussing on the desired differences. If the projections of both samples have small values for a dimension, those small values contribute to a small distance between two samples, yielding a high similarity between the samples. If the projections of both samples have similarly large values for a dimension, these can contribute to a small distance between two samples - but only if they are similarly large. This is only the case if both samples have a similarly large deviation from the timestamps of the model sequence, which is only the case, if they show a similar temporal behavior. If the projections of both samples have differing values for a dimension, these values increase the distance between both samples, emphasizing their inter-sample difference for this dimension.

By using the interpretation methods described in Chapter 4.6.2, Γ_{ST} also allows inspecting, which dimensions are of highest importance for the classification. This allows a better interpretation of the classification results, and also enables a knowledge transfer into potentially faster rule-based algorithms. Since the Γ_{ST} components also encode the temporal position of the relevant n-grams, this can also be used to synchronize the data with existing radio frequency (RF) data, which itself can lead to its own set of failures (e.g. coverage or interference failures), caused for example by low signals of the reception level (RXLEV), the reception quality (RXQUAL) or the received signal code power (RSCP).

Limitations of Γ_{ST} Features

One *limitation* of Γ_{ST} is the high dimensionality of the resulting feature space. To address this issue and to simplify the subsequent projection, classifier training and application, we reduce the final feature space by utilizing the redundancy between the S_{2C} and S_{MOC}, and the S_{2C} and S_{MTC} feature vectors, once they are projected with the structural $\delta - n$ matching of $\hat{\Phi}(s, s^M, \hat{s}^M, \delta, n)$. Table 5.4 shows the binarized $\hat{\Phi}$ features of a snippet of the complete and the reduced feature vector.

Before dimensionality reduction	
s_{2C}	`011110110101101101110101111011`
s_{MOC}	`011110100000000001110101100000`
s_{MTC}	`000000010101101100000000011011`

After dimensionality reduction	
s_{2C}	`0111101101-11011-1110101111-11`
s_{MOC}	`01---10-----01---110---`
s_{MTC}	`----010----10-----011--`

Table 5.4: Exemplary snippet of the binarized $\hat{\Phi}$-projections of a sequence s before and after dimensionality reduction

The dimensionality reduction is achieved by first removing single client dimensions of the currently inactive client which are always zero, i.e. s_{MOC}-dimensions whenever the MTC client is active, and vice versa. Furthermore those representations of dimensions are removed which are redundant between the single client and the multi-client vector, i.e. between s_{MOC} and s_{2C}, and s_{MTC} and s_{2C}, keeping only the s_{2C} dimension. Since the $\hat{\Phi}$ serve as the basis for the Φ features, these can simply be projected onto the new, reduced representation afterwards, which, when concatenated, constitute the final Γ_{ST} feature vector. To further reduce the dimensionality of Γ_{ST}, one could also select features that are most relevant for the classification task, e.g. by applying efficient feature selection methods like RDE [BBM08].

Another related *limitation* of Γ_{ST} exists in form of a potentially problematic class-wise feature balancing. If S^M contains an unbalanced number of samples per sequence class, this will lead to many, potentially redundant features for over represented data aspects (e.g. class specifics), while other class- or data aspects could remain nearly uncovered, due to an insufficient number of s^M representing these aspects. This makes multi-class learning harder, because these over represented features might outweigh less represented features and may therefore produce results prone to classify the corresponding class. To address this issue and to further improve the expressiveness of the resulting Γ_{ST} features, the sequences for S^M should be carefully selected, to properly and equally represent all classes. While we made sure not to use duplicate sequences in any of our data sets, we did not include such an additional sequence selection optimization.

5.5 System Layout

This section will introduce the actual system for the detection and prediction of classes of sequence behavior and the utilized learning methods. The system description will be kept as abstract as possible, to allow an application to other relevant use cases. Figure 5.1 shows the training phase of the system, in which the sets of sequences S are used to create the feature vectors, which are then used to train the required classifiers, responsible for the detection and prediction of properly represented model classes.

5.5.1 Model Class Detection and Prediction

In our use case, the classes of sequence behavior are defined by the different ways communication sequences between both clients can fail. When samples of failed sequences of a new data campaign need to be classified, we could assume a hypothetical scenario in which no new failure classes are found in the new campaign, i.e. all potential failure classes have already been seen before. However, this is not true in practice, where new types of previously unseen failures occur indeed. This is also true in our data sets, where only a limited number of failures classes have a sufficient size to properly evaluate supervised classification *models* with them. We

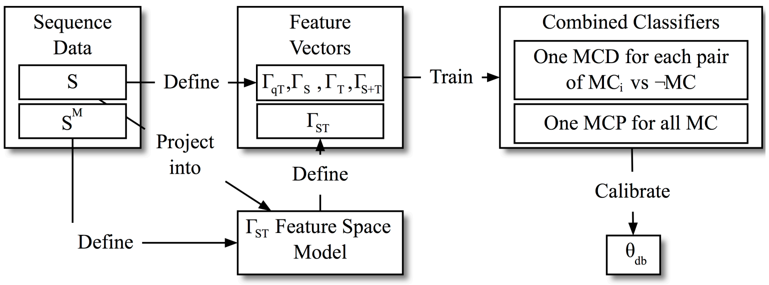


Figure 5.1: Training of the detection and prediction system

denote such classes as *model classes*, or MC. Sequences of *Other Failures*, i.e. of insufficiently large failure classes are denoted as *non model classes*, or $\neg MC$.

As a consequence we design our system to contain two major components, which allow to detect whether a new sample is potentially an MC sample, and if that is the case, to predict the model class. Accordingly, those steps are called the model class detection (MCD) and the model class prediction (MCP). For the MCP a classifier is trained in a multi class approach, learning to discriminate only samples of the MC, but not the $\neg MC$. For the MCD multiple classifiers are trained, one for each MC. Each of those classifiers is trained in a two class approach, learning to discriminate the respective MC against samples of $\neg MC$. After training the MCP and MCD classifiers, we can predict the failure class of a new sample by predicting a MC with the MCP classifier, and then using the MCD classifier trained for this MC to confirm or reject this prediction.

5.5.2 Learning Methods

While unsupervised or semi-supervised methods have shown to produce good results on textual and structural data and could be relevant to our problem of model class detection, supervised methods are regularly outperforming them and are the preferred solution, if labeled training data is available. As discussed in Section 5.2, Decision Trees, Markov Classifiers and LSTM are learning methods widely used in the domain of process mining, while MLP and SVM are more widely used on non-sequential data representations of sequential data. For these reasons we are conducting our evaluations on those methods, to provide a broad picture of the classification performances achievable on the discusses sequential (Γ_{qT}), non-sequential (Γ_T, Γ_S, Γ_{S+T}) and semi-sequential (Γ_{ST}) feature representations. We also include the classification performance using k-nearest neighbors to provide an additional baseline. All of the models below have been chosen using standard cross-validation based model selection.

Decision Tree

Decision trees model training data based on their sequence, where their shared prefix paths build the root of the tree, which branches along the sequence down to the leafs, annotating the transitions with their respective probabilities. They are widely used, specifically in process mining in [MDFDG14, CLF$^+$14, LSD$^+$15, ULD16] and as random forest of decision trees[LCDF$^+$15]. We use them as classifier, based on the sequential Γ_{qT} features, using additional suffix padding to achieve equivalent length sequences.

Markov Classifier

Markov Models are commonly used in process mining [PNC11, LGN12, LSD$^+$15, ULD16], where they are primarily used for predicting objectives like the remaining time or the next event, and not for sequence classification. It can also be used for classification though, as Markov Models represent the data of each class in the training data as a Markov process. This allows calculating the class-wise path probabilities for a new sequence, and predicting the most probable class by the highest overall path probability. We apply such a classifier on the sequential data representations of the Γ_{qT} feature spaces.

LSTM RNN

Recurrent Neural Networks with Long Short-Term Memory nodes are a type of classifier recently used e.g. in process mining [TVLRD17, ERF16]. Deep Learning approaches work best with large training data sets. In our use case, getting a large amount of labeled samples is not easy, thus a deep learning approach might not be the best way to address this problem. However, recurrent neural networks (RNN) with long short-term memory (LSTM) units have shown great performance on sequence prediction problems, s.t. evaluating their performance on this problem is still highly interesting. For our experiments we are using the tensorflow [AAB$^+$15] implementation of RNN with LSTM. The Γ_{qT} features are specifically designed with an LSTM RNN in mind. To achieve samples of homogenous length per batch, we added empty events to the end of each sequence. Since the history of each event is of specific relevance in the event sequence handling in LSTMs, this suffix-padding is a good solution, as it allows to assure that the starting events are not empty. In our experiments we achieve the best results when using one-hot label encoding, a single hidden layer of 20 nodes, 300 epochs and a batch size of 10. Further details on the concepts and ideas behind LSTMs, as well as their formal definitions, are provided in Chapter 7.1.4.

KNN

K-nearest neighbor classifiers are classical distance-based baseline classifiers from the field of natural language processing. We achieve the best results with a value of $k = 5$, using the euclidean distance while additionally weighing points by the inverse of their distance, s.t. closer points have a larger impact.

MLP

Multi Layer Perceptrons are a widely used type of neural network sequence classifier, e.g. in [LWR$^+$10, MLI$^+$15]. We achieve the best results by using a single hidden layer with 80 nodes and the identity function as activation function. Further explanations on the reasoning behind MLP classifiers are provided in Chapter 7.1.3.

SVM

Support Vector Machines are a supervised learning methods, training a maximal margin separating hyperplane between linearly separable class data. While this can also be extended to non-linearly separable class data, we are using a linear kernel, which has shown very good results given sufficiently high-dimensional data, and specifically for protocol-based communication data [ORLS14, PAF$^+$09, WPS06, WS04]. For the MCP evaluation we are using a one-vs-rest (OVR) approach, as this includes calculating a separating hyperplane for each model class MC, which allows a confidence calibration to optimize the system precision, as explained in the next section. Further details on the optimization problem of 2C-SVM, its decision functions and the use of kernels are provided in Chapter 7.1.1.

5.5.3 Combined Classification System

We combine different classifiers to separate each MC from $\neg MC$ samples (via the MCD), and to predict the correct MC (via the MCP). The corresponding predictions are obtained by applying their respective prediction functions to the feature vector $\Gamma(s)$ of a test sample s, using one of the previously defined feature spaces, i.e. $\Gamma \in \{\Gamma_{qT}, \Gamma_T, \Gamma_S, \Gamma_{S+T}, \Gamma_{ST}\}$. The function for the model class prediction classifier is F_{MCP}, with

$$y_{MCP} = F_{MCP}(\Gamma(s))$$

and with $y_{MCP} \in \{MC_1, ..., MC_k\}$, the set of all k model class labels. The prediction function for the model class detector is $F_{MCD}(MC_i)$, obtaining the prediction of the classifier trained for MC_i via

$$y_{MCD} = F_{MCD}(MC_i, \Gamma(s))$$

with $y_{MCD} \in \{MC_i, \neg MC\}$. As one can see, F_{MCP} returns one of the MC-labels, and $F_{MCD}(MC_i)$ returns the label of the class MC_i, or $\neg MC$.

We are combining these prediction functions by using two confidence ratings as a way to ensure a higher confidence in the predictions of the combined (MCP and MCD) classifier, as this helps largely improving the overall classification precision of our approach, which is crucial in the practical application. These confidence ratings produce the binary results *High* and *Low*. They can be logically combined and can be interpreted either as providing support for the prediction of the MCP (*High* confidence) or objecting against its prediction (*Low* confidence).

Since the output of F_{MCD} is limited to a single MC_i and $\neg MC$, it is used as the first confidence rating for y_{MCP}, answering the question of whether the provided y_{MCP} really belongs to MC_i, or whether it belongs to $\neg MC$. For this purpose, the confidence rating function $\Theta_{MCD}(\Gamma(s)) \in \{\textit{High}, \textit{Low}\}$ is defined as:

$$\Theta_{MCD}(\Gamma(s)) = \begin{cases} \textit{High} & \text{if } F_{MCD}(y_{MCP}, \Gamma(s)) = y_{MCP} \\ \textit{Low} & \text{else} \end{cases}$$

To obtain further confidence on the classification result of F_{MCP}, we define an additional confidence rating Θ_{db}. It uses the decision boundary of the MCP classifier of the predicted class. The idea behind Θ_{db} is to calibrate the decision boundary of each MCP classifier towards more conservative values, requiring a sample with $y_{MCP} = MC_i$ to cross a stricter decision boundary for this MC_i to obtain a *High* confidence for Θ_{db}, reducing the false positive rate and increasing the precision. As it is defined over the decision scores, it can be applied to any classifier which provides access to its decision scores or probabilities. For this purpose we need an additional function for accessing the prediction score of the MCP via the function $D_{MCP}(\Gamma(s))$. We also require the existing bias b of the MCP classifier of y_{MCP}, and a parameter $\theta_{DB} \geq 0$ to define the new bias $b_{db} = b + (D_{\varnothing} - b) \cdot \theta_{db}$, with $D_{\varnothing}$ representing the mean of those decision scores of the training samples that have been correctly classified by the MCP. As such, the parameter θ_{db} gives rise to the following definition of the confidence rating Θ_{db}:

$$\Theta_{db}(\Gamma(s), \theta_{db}) = \begin{cases} \textit{High} & \text{if } D_{MCP}(\Gamma(s)) + b_{db} > 0 \\ \textit{Low} & \text{else} \end{cases}$$

For example in the two-class definition of the SVM the decision function (as defined in Formula 7.1 in Chapter 7.1.1) is $F_{MCP}(\Gamma(s)) = sign(\langle w, \Gamma(s) \rangle + b)$, with w being the weight vector of the trained model. Here we achieve a positive prediction if $\langle w, \Gamma(s) \rangle + b > 0$. The respective function to access the score is then $D_{MCP}(\Gamma(s)) = \langle w, \Gamma(s) \rangle + b$.

θ_{db} can be changed dynamically during model selection, which allows for a calibration of the model precision, similar to other methods of false positive calibration as used e.g. [SBKR12]. The confidence ratings are then processed together with the prediction y_{MCP} to produce the final predicted label $F_{combined}(\Gamma(s)) \in \{MC_1, ..., MC_k, \neg MC\}$, defined as follows:

$$F_{combined}(\Gamma(s)) = \begin{cases} F_{MCP}(\Gamma(s)) & \text{if } \Theta_{MCD}(\Gamma(s)) = \textit{High} \\ & \wedge\, \Theta_{db}(\Gamma(s), \theta_{db}) = \textit{High} \\ \neg MC & \text{else} \end{cases}$$

The effect of the confidence ratings and the consequently created *High* - confidence prediction subset on the applied evaluation metrics will be elaborated in the next section. Now that the system is trained, i.e. the Γ_{ST} feature space model is defined and the MCD and MCPs are trained and calibrated, we can apply the system to classify unlabeled sequences, as illustrated in Figure 5.2. As just described, the final prediction $F_{combined}(\Gamma(s))$ of this combined classifier for a given sequence s only provides the label of the predicted MC_i if $\Theta_{MCD}(\Gamma(s)) =$ *High* and $\Theta_{db}(\Gamma(s), \theta_{db}) =$ *High*, and $\neg MC$ otherwise. This, together with the still accessible Θ-confidence ratings allows for an effective filtering of reliable and unreliable predictions, which is highly relevant in the practical application.

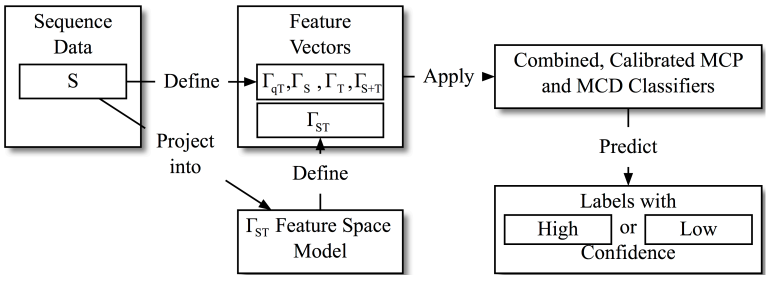


Figure 5.2: Application of the detection and prediction system

System Limitations

One *limitation* of this combined system is the currently unused opportunity of a closer integration of the currently separated MCP and MCD steps during the training and the application phase. This provides an opportunity to further increase the system classification performance, as both steps could complement each other and provide class information currently not available to the respective other. Another *limitation* is the requirement to define Θ_{db} via model selection, which could be problematic given the potentially small data sets available.

5.6 Evaluation

In this section we start with describing the parameters and settings used for the feature spaces and classifiers during the evaluation, as well as the utilized metrics. Afterwards we evaluate the model class prediction (MCP) component of the proposed system. For this purpose we evaluate the Γ_{qT} feature space with LSTM classifiers, and the Γ_T, Γ_S, Γ_{S+T} and Γ_{ST} feature spaces with the MLP, KNN and SVM classifiers. We also conduct additional comparative analyses on various statistical properties of the feature spaces, allowing for more detailed conclusions on their individual advantages and disadvantages. Finally we select the best performing combinations and evaluate their model class detection (MCD) and combined

MCP and MCD classification performance, simulating the complete system workflow. For all evaluations we used 20 times random sampling, each with a 5 times cross validation. For a proper evaluation we made sure all conducted comparisons between classifiers and feature spaces were done on the same respective samplings. For the event n-gram sizes of the Γ_S, Γ_{S+T} and Γ_{ST} feature spaces we used a fixed n-gram size of $n = 2$, which yielded the overall best results. We also tested using multiple values of n simultaneously, as e.g. described in [LXLZ15]. However, the performance increase was only minimal. As a convention, all results are listed as the mean and standard deviation in percent.

5.6.1 Evaluation Metrics

The general system application of reliably detecting and predicting known amidst unknown sequence classes, and its concrete practical application in failure classification enforces a focus on two primary objectives: (1) to obtain reliable predictions (2) for as many MC samples as possible. Precision, recall and the F1 score capture those aspects. For the evaluation of the MCP and the MCD classifiers they are calculated for multiple cross validation repetitions, s.t. we chose to calculate their unweighted class mean (denoted with the keyword *macro*), because we already configured the sampling procedure to produce similarly sized model classes. Whenever the $\neg MC$ class participated in the evaluation (in MCD and combined classifiers) we excluded it from the calculation of the classwise mean values, because our focus is on the correct detection of MC samples. By combining MCP and MCD classifiers and applying confidence ratings, we effectively create a filter, allowing us to focus solely on the created *High*-confidence subset of the predictions, expected to contain only those samples and labels which truly are of a model class MC and are correctly classified as such. Consequently we also have to adapt our metrics. The set P of the positive samples is defined as the set of samples contained in the *High*-confidence subset, which is split into the subsets TP and FP. TP consists of the correctly predicted samples of MC in this subset, and FP consists of falsely predicted samples of MC and also samples of $\neg MC$ in this subset. The set N of the negative samples is defined as the set of samples in the *Low*-confidence subset and is split into the sets TN and FN. TN consists of samples of $\neg MC$ and falsely predicted MC, and FN consists of samples of correctly predicted samples of MC. Based on these values precision and recall are defined as usual, with $precision = TP/P$ and $recall = TP/(TP + FN)$. However, to correctly address objective (2) we have to calculate an additional *effective recall* by considering all existing MC samples, not only those in the *High*-confidence subset. Therefore we define the effective recall as the recall of correctly predicted samples of MC over the sum of samples in all MC, i.e. *effective recall* $= TP/\sum s_{\#MC_i}, \forall MC_i$. Together with the precision over the *High*-confidence subset, this metric allows for a conclusive analysis of the overall system classification performance, as provided by the combined classification processing.

5.6.2 Experiments on MFC data

To be able to apply the selected learning methods, we have to assure sufficiently sized failure classes. To obtain any model classes at all, we sample only from classes with a minimum size of 15 sequences. To improve the interpretability of our classification results we opted for similarly sized failure classes, which we achieved by limiting the size of each failure class to a maximum of 25 sequences. As such, the number of samples per MC used for the evaluation, $s_{\#MC}$, is $15 < s_{\#MC} \leq 25$. To simulate the complete system, we also need members of the *Other Failures* class, of which we used all 86 available sequences, i.e. $s_{\#\neg MC} = 86$. In the MCD evaluation this allows highlighting the *detection* purpose of the method, as the ratio of samples of MC_i to $\neg MC$ is approximately $1 : 4$. In the combined evaluation the ratio of all MC samples to $\neg MC$ samples is approximately $1 : 1$, which helps in the interpretation of the classification results. For defining the Γ_{ST} features we used all 6,077 labeled and unlabeled MFC samples as model sequences S^M. After applying an additional redundancy-based dimensionality reduction, this resulted in a feature space of 294,435 dimensions. As previously described this use of the unlabeled sequences helps to extract additional information about the behavior of the projected sequence.

Evaluation of the individual MCP and MCD classifiers

The purpose of the first set of experiments is to find the best performing learning method for all feature spaces, s.t. we can restrict the further experiments to this learning method, allowing to focus on the feature space analyses. Since Γ_{ST} is further parametrized by δ, we start with analyzing the impact of different values of δ on the MCP classification performance of Γ_{ST} using the SVM classifier. The mean sample length in the MFC data set is 44.11 events, with a standard deviation of 13.51 events. Since δ encodes the positional variance of the matched n-grams within the projected sequence, it does not make sense to increase its size beyond a value of $\delta = 60$, at which the complete average sequence length is covered. Since we expect a high importance of a similar positioning of the matched n-grams, we expect better results for smaller values of δ. The left plot in Figure 5.3 shows the results for $\delta \in [0, 60]$. As expected we achieve the best results with a value of $\delta \leq 10$. For that reason we focused stronger on the range of $\delta \in [0, 10]$, which is illustrated in the right plot in Figure 5.3, allowing to further reduce the selection of an optimal value down to $\delta = 5$. As this value allows a positional variance of ± 5 events on S^M, it can additionally be explained by the circumstance that event sub-sequences, which are crucial to the protocol, like the call setup (also illustrated in Table 5.2) take around 10 events in the s_{2C} representation, requiring any sequence to match the contained events.

Now that we know a proper setting of δ we can conduct a comparative evaluation of all MCP classifiers on the MFC data set, using all feature spaces. The results are shown in Table 5.5. Of the process mining methods applied on Γ_{qT} and

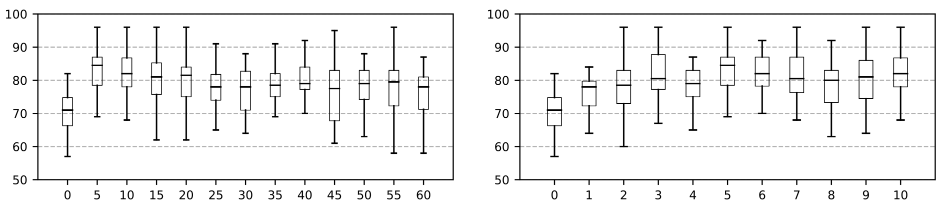


Figure 5.3: MCP evaluation on MFC data for Γ_{ST}: F1 scores (*macro*) in % for ranges of $\delta \in [0, 60]$ (left) and $\delta \in [0, 10]$ (right)

Γ_{qT^*}, the decision tree showed the overall best performance, specifically on Γ_{qT^*}, i.e. the original sequence representation. However, both the Markov and the LSTM classifier achieve an improved performance on the temporally enriched Γ_{qT} feature space. When compared to the performance on the other feature spaces though, all of those approaches commonly used in the field of process mining are clearly outperformed by the other feature spaces and learning methods. While this was to be expected, given that the process mining approach, and specifically deep learning approaches like LSTM usually require much larger training data sets, also the lack of the additionally included concrete temporal information is highly relevant, since they are even outperformed by Γ_T, which only contains strongly reduced structural information about the data. Hence the SVM classifier performs best on all feature spaces, outperforming the otherwise widely used MLP, as well as (obviously) the KNN approach. For those reasons we are using it for the remaining experiments. The results of the SVM classifier also show, that in the optimal scenario in which all MC are known, good results can already be achieved without using the proposed θ_{db} system calibration.

The results of the MCD evaluation using the SVM classifier are shown at the bottom of Table 5.5. Obviously discriminating the MC and $\neg MC$ is harder than separating the MC in the MCP setting, which is to be expected, as the $\neg MC$ samples are very heterogenous. However, using semi-supervised learning via a One-Class SVM to model each MC_i against the $\neg MC$ performed even worse. Of all feature spaces Γ_{S+T} performs best, while the specifically crafted Γ_{ST} feature space is slightly outperformed by all other feature spaces. While one might think that Γ_{ST} does not look that promising yet, the combined classifier evaluation will show, that it performs better than its competitors in the final system layout, when the effective recall becomes relevant.

Evaluation of the combined classifiers

Now that we established some understanding of the performance of the individual MCD and MCP classifiers, we will now evaluate for each qualified feature space its combined classification performance, also integrating the previously described confidence ratings. We do this to find the feature space which has the highest precision, at the highest possible effective recall, which is highly relevant for an

MCP		F1 Score (*macro*)	Precision (*macro*)
Γ_{qT^*}	Markov	30.90 ± 8.06	34.62 ± 10.97
	DCT	52.58 ± 10.12	56.03 ± 11.26
	LSTM	45.36 ± 11.78	49.25 ± 13.42
Γ_{qT}	Markov	33.57 ± 6.91	34.22 ± 9.18
	DCT	48.89 ± 9.74	52.45 ± 11.44
	LSTM	47.38 ± 7.85	52.55 ± 8.14
Γ_T	KNN	77.57 ±9.05	80.27 ±9.13
	MLP	80.94 ±6.83	83.55 ±6.98
	<u>SVM</u>	84.72 ±7.34	87.33 ±6.65
Γ_S	KNN	77.51 ±8.89	81.79 ±7.96
	MLP	83.07 ±7.00	85.58 ±6.41
	<u>SVM</u>	83.60 ±9.01	86.17 ±8.51
Γ_{S+T}	KNN	79.10 ±8.14	82.44 ±7.89
	MLP	84.27 ±7.81	87.08 ±7.24
	<u>SVM</u>	85.25 ±7.17	87.67 ±6.83
Γ_{ST}	KNN	77.67 ±8.40	79.93 ±8.31
	MLP	78.53 ±8.31	82.03 ±7.55
	<u>SVM</u>	83.67 ±7.13	85.70 ±6.77

MCD		F1 Score (*macro*)	Precision (*macro*)
Γ_T	SVM	62.91 ±19.43	64.54 ±20.33
Γ_S		63.33 ±21.72	70.68 ±23.85
Γ_{S+T}		65.53 ±20.64	72.57 ±22.21
Γ_{ST}		58.63 ±25.17	76.72 ±28.09

Table 5.5: Results of the individual MCP and MCD evaluations on MFC data (%)

effective system in the practical application. Achieving high precision predictions means that we can trust the results to be correctly classified and to not contain any samples of $\neg MC$ falsely being classified as an MC sample. And getting the high effective recall means, we get this predictive behavior for a larger portion of the MC samples that are actually contained in the test set. This aspect is illustrated in Figure 5.4, which shows the percentage of *High*-confidence samples of MC to all samples of MC in the test set, under a shifting parameter $\theta_{db} \in [0, 1.0]$. The more θ_{db} is increased, the less samples of MC are actually contained in the *High*-confidence subset, reducing the potential effective recall.

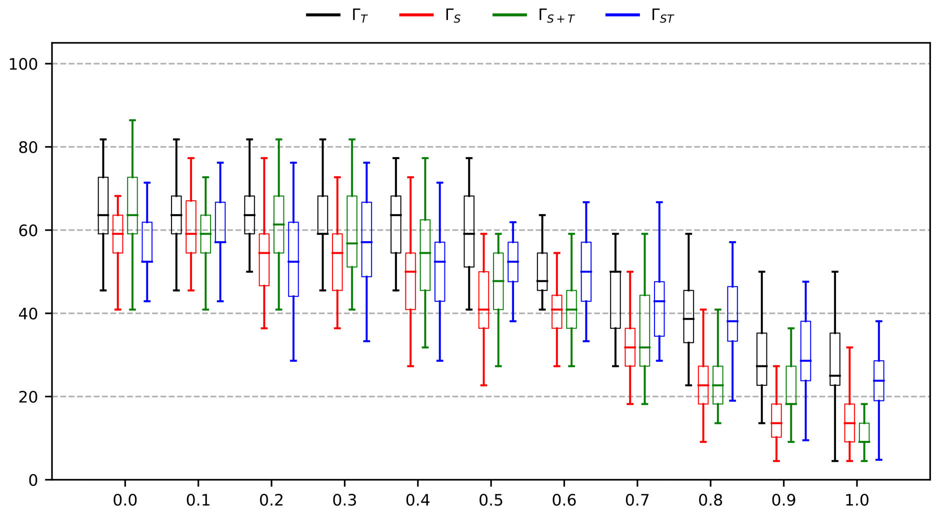


Figure 5.4: Percentage of MC samples in the *High*-confidence set, for $\theta_{db} \in [0, 1.0]$, for Γ_T, Γ_S, Γ_{S+T} and Γ_{ST}

For comparing the combined classification performance of the different feature spaces with the SVM classifier, we need to select values of θ_{db} representing practically relevant values of precision and recall, which are similar for the respective feature spaces. Figure 5.5 contains the precision, recall and the effective recall of the classification results for $\theta_{db} \in [0, 1.0]$. The precision starts to reach 100% for most classifiers at $\theta_{db} = 0.8$. At this value also the recall reaches the maximum of 100%. When looking at the concrete values of mean and standard deviation, shown in Table 5.6, we see that the recall can not be further increased, and that the corresponding precision can be selected s.t. it is around 93% for Γ_{ST}, Γ_{S+T} and Γ_S. Since further increasing the precision would not further increase the recall, and 93% is already a reasonable system precision, we will use this value and the respective settings of θ_{db} for the further analyses. Thus we are using $\theta_{db} = 0.8$ for Γ_S and Γ_{S+T}, and $\theta_{db} = 0.9$ for Γ_{ST}. In this respect, the performance of Γ_T was not sufficiently high to achieve similar values of precision and recall, which is why we used $\theta_{db} = 1.1$ there, achieving relatively close values for further analyses.

Now we need to evaluate which feature space offers the best effective recall,

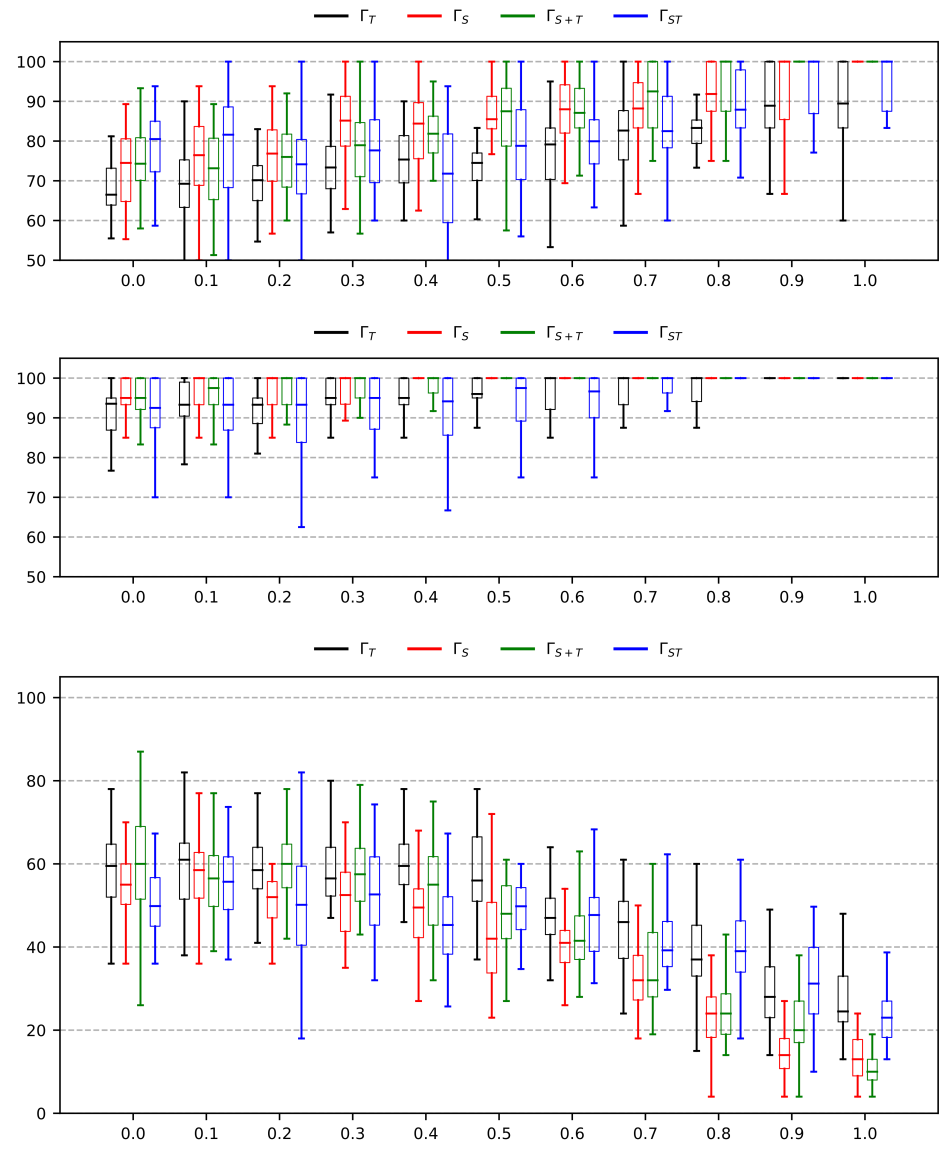


Figure 5.5: Combined classification results on MFC data: precision, recall, effective recall (from top to bottom) in % for $\theta_{db} \in [0, 1.0]$, for Γ_T, Γ_S, Γ_{S+T} and Γ_{ST}

i.e. the fraction of MC samples that can be recovered from the set of test samples, with those high values of precision and recall previously described. At their respective values of θ_{db}, Γ_S has the lowest effective recall of 25.05%, followed by Γ_{S+T} with 26.27%, and Γ_{ST} with an effective recall of 31.79%. Thus Γ_{ST} produces a precision performance similar to both Γ_{S+T} and Γ_S, while achieving a 5.5% higher effective recall as Γ_{S+T}, and a 6.7% higher effective recall as Γ_S. This means, we can get for 31.79% of the MC samples in the test set a correct prediction with 93.11% precision and 100% recall when using Γ_{ST}, compared to 26.27% effective recall, 93.51% precision and 100% recall when using Γ_{S+T}, and even worse when using Γ_S. These results are highly relevant in practice, as they effectively allow filtering near-certain from uncertain predictions with a very high precision. These results are also highlighting the effectiveness of both combined feature spaces, with a significant advantage for the Γ_{ST} feature space. In that regard both structural-temporal feature spaces Γ_{S+T} and Γ_{ST} outperform Γ_S and Γ_T: Whereas Γ_T has a relatively good effective recall, but a relatively low precision, Γ_S has an acceptable precision, but a low effective recall. This renders both feature spaces less practically relevant than their combined counter parts, highlighting the relevance of combined structural-temporal feature spaces.

The dimensions most relevant for the respective classification results in this use case were security handshake events, followed by the existence of events representing a successful response to the most relevant key protocol states, like a successful radio bearer setup. As we saw in the MCP evaluation, the temporal features are also relevant and utilized in both combined feature spaces. The Γ_{ST} feature space also has an advantage here, as its S^M-based feature space allows locating the concrete positions and structural-temporal properties of the relevant events within the sequence, which are in the evaluation often identified as responses occurring too late in the sequence, or security mode negotiations at anomalous sequence positions. All of this can then be used to obtain deeper insights into the data, which can help manual analysts to limit the number of causes for this specific failure class.

Due to their potential in combining classifiers of different features spaces, we also evaluate the classification performance of an ensemble method [Die00], namely the ensemble classifier $\mathcal{E}$, which could potentially further optimize precision and effective recall. It predicts the combined classification results by using a majority voting over the predicted labels of Γ_S, Γ_{S+T} and Γ_{ST}. Table 5.6 shows its results when using their default trained models of $\theta_{db} = 0.0$, and for their optimized decision boundary models, using $\theta_{db} = 0.8$ for Γ_S and Γ_{S+T}, and $\theta_{db} = 0.9$ for Γ_{ST} respectively. Γ_T has not been used due to its lower performance. When using the default models at $\theta_{db} = 0.0$, the ensemble classifier in fact achieves the best precision at the cost of the effective recall, an effect similar to the trade-off of θ_{db}. For the optimized models of $\theta_{db} \in \{0.8, 0.9\}$ the ensemble classifier achieved worse results though.

	θ_{db}	F1 Score	Precision	Recall	Eff. Recall
Γ_T	0.0	76.37 ±7.90	69.09 ±8.22	91.12 ±7.73	61.57 ±9.39
	1.1	90.71 ±13.28	87.40 ±16.75	98.44 ±7.67	17.25 ±7.49
Γ_S	0.0	80.82 ±7.18	73.74 ±8.42	95.32 ±5.55	57.94 ±10.17
	0.8	95.01 ±6.26	92.65 ±8.92	99.90 ±1.00	25.05 ±7.98
Γ_{S+T}	0.0	81.23 ±7.65	75.28 ±8.09	95.02 ±5.54	60.50 ±12.77
	0.8	95.64 ±5.54	93.51 ±8.13	100.0 ±0.0	26.27 ±8.64
	0.9	97.36 ±4.86	95.97 ±7.39	100.0 ±0.0	20.38 ±8.09
Γ_{ST}	0.0	81.52 ±8.83	77.74 ±10.97	91.73 ±7.35	50.84 ±8.28
	0.8	91.72 ±8.29	88.29 ±10.34	99.17 ±4.49	39.88 ±8.81
	0.9	95.33 ±6.32	93.11 ±9.16	100.0 ±0.0	31.79 ±10.33
$\mathcal{E}$	0.0	81.98 ±8.33	78.53 ±9.07	90.10 ±9.40	55.78 ±10.73
	0.8	88.74 ±13.49	89.76 ±13.65	90.72 ±13.58	23.59 ±7.76
	0.9				

Table 5.6: Results of combined classification evaluation on MFC data (in %)

Significance analysis

To further substantiate the results of the previous section, we conduct significance tests on both of our theoretical hypotheses, namely that the combined feature spaces Γ_{S+T} and Γ_{ST} outperform the base feature spaces Γ_T and Γ_S in terms of effective recall at similar precision (Hypothesis H^A), and that the more complex combined feature space Γ_{ST} outperforms the simpler combined feature space Γ_{S+T} under the same premises (Hypothesis H^B). For the formulation of the hypotheses we denote θ_{er} as the minimal effective recall.

For hypothesis H^A the null hypothesis H_0^A is defined as follows: When using Γ_S or Γ_T for achieving a test set precision mean of 93%, a fraction of p_0 samplings have an effective recall $\geq \theta_{er}$. The alternative hypothesis H_1^A is then defined as follows: When using Γ_{ST} or Γ_{S+T} for achieving a test set precision mean of 93%, a fraction of $\hat{p}$ samplings have an effective recall $\geq \theta_{er}$. Now we can formulate the question for hypothesis H^A: Is there sufficient evidence at the $\alpha = 0.05$ level to conclude that the effective recall for the high precision classification performance is increased, when using one of the combined feature spaces Γ_{ST} or Γ_{S+T} instead of one of the individual feature spaces Γ_T or Γ_S? And at which minimal effective recall θ_{er} does this hold? The results for the minimal θ_{er}, at which we can reject H_0^A in favor of H_1^A (i.e. above which $p \leq \alpha$ always holds for the resulting p-values) are shown in Table 5.7, for each pair of base and combined feature space, as calculated on the same sampling that have also been used for the previous combined MFC evaluation. We can see that H_0^A can be rejected for Γ_T for values of $\theta_{er} \geq 5\%$, i.e. for nearly all values of θ_{er}, excluding those which do not occur in the combined feature spaces due to their generally higher effective recall. For Γ_S, H_0^A can be rejected for $\theta_{er} \geq 9\%$ for Γ_{ST}, and for $\theta_{er} \geq 14\%$ for

Γ_{S+T}. This means Γ_{ST} is better for a larger number of samplings, while Γ_{S+T} starts outperforming Γ_S later - both of which is also relevant for hypothesis H^B.

		Γ_{S+T}	Γ_{ST}
H^A	Γ_T	5%	5%
	Γ_S	14%	9%
H^B	Γ_{S+T}	-	30%

Table 5.7: Values of minimal θ_{er} required to reject hypotheses H_0^A and H_0^B at $\alpha = 0.05$

For hypothesis H^B the null hypothesis H_0^B is defined as follows: When using Γ_{S+T} for achieving a test set precision mean of 93%, a fraction of p_0 samplings have an effective recall $\geq \theta_{er}$. The alternative hypothesis H_1^B is then defined analogous: When using Γ_{ST} for achieving a test set precision mean of 93%, a fraction of $\hat{p}$ samplings have an effective recall $\geq \theta_{er}$. The question for hypothesis H^B is then: Is there sufficient evidence at the $\alpha = 0.05$ level to conclude that the effective recall for the high precision classification performance is increased, when using the complex combined feature space Γ_{ST} instead of the simpler combined feature space Γ_{S+T}? And at which minimal effective recall θ_{er} does this hold? As shown in Table 5.7, H_0^B can be rejected for all values of $\theta_{er} \geq 30\%$, showing that Γ_{ST} indeed outperforms Γ_{S+T}, a fact that is further strengthened by the performance advantage of Γ_{ST} over Γ_{S+T}, as shown for hypothesis H^A.

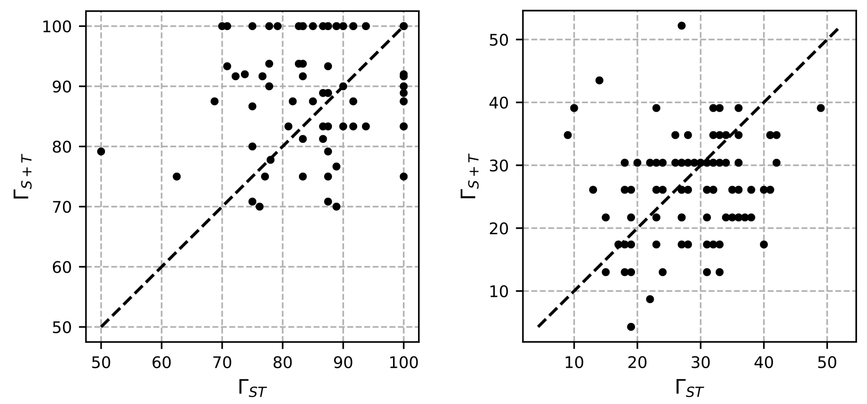


Figure 5.6: Precision (left) and Effective Recall (right) of Γ_{ST} against Γ_{S+T} (in %)

We will now further elaborate hypothesis H^B by analyzing the distribution of precision and effective recall, when using either feature space on the same test sets. The results of this performance variance analysis are shown in Figure 5.6. As previously stated, and shown in Table 5.6, we conducted the significance tests on

SVM classification models calibrated for an average precision $\geq 93\%$. As shown in the left plot of Figure 5.6, the models of both Γ_{S+T} and Γ_{ST} show similar performance distributions, with a slight advantage for Γ_{S+T}, due to its slightly higher average precision of 93.51%, compared to the 93.11% of Γ_{ST}. However, as the right plot shows, the effective recall on the same test sets is much balanced towards Γ_{ST}, clearly supporting hypothesis H^B. As a result these analyses support our conclusion that Γ_{ST} is the most advantageous feature space in the discussed use case.

5.6.3 Experiments on AFC data

Due to the already discussed shortcomings of the AFC data set properties, we are only interested to see, whether the MCP are capable of discriminating the model classes of the AFC data set at all, and how much of a performance improvement we can expect with a larger training data set, which is possible only on the AFC data set. Similar to the class size restrictions described for the MFC evaluation, we have to ensure sufficiently large as well as similarly sized failure classes. To reflect smaller and larger training data sets, we evaluate two different setups. The first setup is defined with comparability to the MFC evaluations in mind. Hence we use $s_{\#MC} = 25$, resulting in the 13 sufficiently sized failure classes listed in Table 5.1. In the second setup, selecting $s_{\#MC} = 100$ allows for a larger training data set, resulting in 4 sufficiently sized failure classes. For defining the Γ_{ST} features we used all 3,264 sequences as model sequences S^M, resulting in a feature space of 144,938 dimensions after redundancy-based dimensionality reduction.

MCP	$c_{\#MC}$	F1 Score (*macro*)	Precision (*macro*)
Γ_{S+T}	25	46.84 ±4.71	49.17 ±6.17
Γ_{ST}		42.00 ±5.63	43.97 ±6.12
Γ_{S+T}	100	71.15 ±4.09	72.08 ±4.17
Γ_{ST}		62.90 ±6.05	63.67 ±6.08

Table 5.8: Results of the individual MCP evaluations on AFC data (in %)

Table 5.8 shows the results of the MCP evaluations on the AFC data set. Due to the differences in the MFC and the AFC data, we expected a worse classification performance than on the MFC data, which indeed occurs. However, for the larger sets of training data with $s_{\#MC} = 100$ the results are largely improved, which shows, that the AFC data set still contains a sufficient number of discriminative features to enable classification. This also documents the potential for an equally increased classification performance on the MFC data, once more class-wise training data is there available as well - which also applied to the general use case of similar classification problems.

5.7 Conclusion

This chapter addresses theoretical and practical issues, relevant when analyzing real-time log data of structural, temporal processes using specific structural-temporal feature spaces, specifically when solving mobile communication failure classification problems. On the theoretical side we present an analysis of structural and temporal data properties, specifically on the discussed format of mobile communication data. We introduce and discuss novel individual and combined feature spaces utilizing those properties to obtain a good data representation. We conduct a comparative performance evaluation of these feature spaces against feature spaces commonly used in related work, on a range of classifiers. We also show in various evaluations and via hypothesis testing that both of our combined temporal structural feature spaces Γ_{S+T} and Γ_{ST} outperform their competition counterparts from the research fields of sequence learning and process mining, and that the novel Γ_{ST} feature space excels in classification performance when being compared to all other approaches, including an ensemble method. We also discussed and applied a solution to address the main *limitation* of the Γ_{ST} features, the high feature space dimensionality.

On the practical side we propose a system for the detection and prediction of classes of pre-defined sequence behavior, applied on the use case of the automatic classification of mobile communication failures using the proposed feature spaces and supervised learning, for which we also show how to maximize its classification precision and effective recall via a calibration procedure. We highlight the importance of properly labeled training data, for which we show that our proposed Γ_{ST} feature space is able achieve a precision of more than 93% while having the advantage of an up to 6.7% higher effective recall than the other feature spaces. These results are highly relevant in practice, as they effectively allow separating reliable from unreliable predictions. And with the higher effective recall more reliable predictions can be obtained, further reducing the costs of otherwise unfeasible manual analysis processes.

As an outlook it would be interesting to evaluate the potential of word vector representations like those of [MCCD13] for corpora of structural-temporal data. This would not necessarily reflect the temporal data aspects, and would also require data sets much larger than currently available. However, the sequential and contextual aspects of the event relations could potentially be covered, which could help improving the interpretability of the internal process relations, as well as the overall classification performances.

Chapter 6

Thesis Conclusion

This thesis proposed and analyzed properties and features of sequential and structural data, as well as learning methods and automated systems for this data, specifically in the domains of network communication and code. These are the three primary aspects of this thesis, whose individual results and contributions we will now summarize, followed by the final outlook.

Automated Systems

For this purpose various highly automated systems were introduced, all of which reduce the necessity for additional manual processing, which is relevant for a wide range of web and communication analysis problems, and provides answers to the thesis question

- What is needed to achieve highly automated systems utilizing these features and learning methods?

Chapter 2 introduced a novel approach for detecting covert and tunneled communication of malware using hierarchical detectors for anomalous communication in features of HTTP requests. This system can be adapted to the web surfing characteristics of individual users and can identify the communication of malicious software, tunnels and backdoors. Chapter 3 investigated creating a combined learning-based system for the detection of malicious JavaScript code and a completely automated system for the collection and analysis of JavaScript code. Chapter 4 proposed a system for the validation of mobile communication sequences, which was competitively evaluated using different data representations based on different analysis methods. Chapter 5 proposes a system for the detection and prediction of classes of pre-defined sequence behavior, applied on the use case of the automatic classification of log-data of mobile communication failures. As the thesis also provides analyses of specific practically relevant topics of the proposed systems, like the impact of re-training procedures, the interpretability of the results, or methods to determine the reliability of the results, the proposed systems allow a good adaptation to practical applications. These practical thesis aspects are summarized by the following contributions:

- Proposal and evaluation of highly *automated systems* for solving specific practically relevant learning problems.
- Proposal of methods and graphical representations to increase the transparence and *interpretability* of the system and the obtained results.

Properties and Features

The thesis proposed and analyzed various forms of features and feature combinations with regard to their applicability to sequential and structured data and its specific problems. Specifically token n-grams were found to be highly applicable, especially when extended with additional temporal properties. To address the research questions

- Which properties are relevant for structural and sequential data of network communication and code?
- Which types of features are best suited to represent those properties - and how are they extracted effectively?

the thesis starts in Chapter 2 with utilizing concrete structural, temporal and statistical features like the length or number of individual HTTP requests, as well as their hierarchical combination. While these implicitly stateless features work well for this use case, they can not easily be generalized, as they directly represent concrete data properties. As a consequence Chapter 3 starts utilizing the more general token n-gram features, which is continued in Chapter 4, where token n-gram features are explicitly defined as stateless features. By additionally introducing a stateful feature representation, this chapter allows discussions and comparisons of these stateless and stateful feature spaces. These analyses found that using stateless feature representations requires a higher computational effort (due to its high dimensionality), but usually slightly outperforms stateful features. However, as their performance gap is not very large, stateful features are still a viable option - especially considering their lower computational requirements. Afterwards their potential to complement each other was evaluated, resulting in the highest classification performance. Furthermore it was shown that using non-proprietary data analysis techniques can enable feature representations nearly as expressive as proprietary ones, widening the applicability of the proposed automatic learning approach. Finally Chapter 5 further extends the analysis of combining stateless and stateful features, by introducing novel feature spaces which allow the inclusion of both structural and temporal data properties. Via hypothesis testing and in a broad evaluation on a range of different learning methods, these feature spaces are shown to perform better than various other relevant feature spaces from the research fields of sequence learning and process mining. These thesis aspects are summarized by the following contributions:

- Analysis of the *properties* of structured, sequential data, specifically of network communication and code.
- Proposal of novel *features* or feature combinations to represent those properties.

Learning Methods and Evaluation

The detection and classification performance of the proposed features was evaluated using a wide range of learning methods, from classical machine learning methods like Markov classifiers [LSD$^+$15] and Bayes classifiers [PSBDL18] over kernel based learning methods like One-Class SVM [SWS$^+$00] and Two-Class SVM [MMR$^+$01, SS02] to neural network classifiers like MLP [Sch15] and LSTM [HS97], all of which are relevant in related research fields, like sequence learning or process mining. During the research Support Vector Machines proved to show the most reliable performance, especially when handling small training data sets, and very high dimensional feature spaces. Thus Chapter 2 uses the One-Class SVM very effectively, achieving very good detection performance and false positive rates. This is further emphasized in Chapter 3, which also achieves very good results. Here it is shown that One-Class SVMs can be further improved by additionally including labeled data of the second class, shifting its interpretation more towards the supervised Two-Class SVM. It was also shown that combining different detectors further improved the system performance. Chapter 4 continues these analyses by relying on supervised learning, showing the advantages of the Two-Class SVMs in the evaluation against competitive methods, as well as the additional advantages of using the trained model for the interpretation of the results. Finally Chapter 5 shows the advantages of Two-Class SVMs on a different objective, against competitive learning methods. In the proposed system these learning methods are combined with Multi-Class SVM detectors, which results in a good classification performance and the added benefit of estimating the reliability of the predicted results. These thesis aspects are summarized by the following contributions:

- Proposal of novel ways of combining *learning methods* to achieve the respective classification and detection objectives.
- In depth *evaluation* of the proposed features, learning methods and systems, competitively compared against approaches used in related research areas like IT security, sequence learning and process learning.

Limitations and Outlook

Both practical and theoretical limitations were found during the course of this thesis' research, but none of them represent insurmountable obstacles. For the practical implementation and evaluation of the complete systems it was often crucial to achieve a close and high-performance integration of the system components. As a result the system proposed in Chapter 3 could not be automated as expected, because the lifetime of the malicious activity and the volatility of the JavaScript attacks required a much closer integrated system - a problem, a practical application could solve with a sufficiently performing implementation. Similarly Chapter 5 required a complex implementation of the projection methods to achieve a highly parallelized processing chain. Also theoretical limitations like the additional model selection parameters and sufficiently discriminative training data (Chapter 4) or an improved feature space dimensionality reduction (Chapter 5) did not represent unsolvable problems.

Apart from these main limitations there is still room for improvement. Combining different feature spaces which represent different data aspects still has a very high potential to create more expressive features. Methods like RDE [BBM08] could help at this point by improving the selection of relevant dimensions of those feature spaces. Furthermore word vector feature representations like those described in [MCCD13] could help improving the results obtained with LSTM neural networks, like those used in Chapter 5. While this combination of feature representation and learning method showed great results on other sequential data, and could potentially cover the sequential and contextual aspects of the occurring event relations, integrating the temporal aspects of the datasets used in this thesis would still require additional research.

Chapter 7

Appendix

7.1 Learning Methods

This section provides further explanations and formal notations on the learning methods most prominently used throughout the thesis.

7.1.1 Support Vector Machines

Support vector machines [MMR+01, SS02] are a supervised learning method based on the idea of finding the maximum margin hyperplane between data distributions of two linear separable, labeled classes, which can also be extended to non-linearly separable class distributions by use of the kernel trick. Given labeled training samples (x_i, y_i) with $i \in [1, m]$ and $y_i \in \{-1, 1\}$ of two classes, each hyperplane separating the classes can be described through an orthonormal vector w and its distance b from the origin. It can thus be described by the linear equation $\langle w, x\rangle + b = 0$. A hyperplane splits the feature space in two parts, with one class on either side. As such one can describe the class label of each training sample as $y_i = sgn(\langle w, x_i\rangle + b)$. During the training phase w and b are optimized s.t. the margin between both classes is maximized. For linear separable data this is done by minimizing $\frac{1}{2}||w||^2$, s.t. the side condition $y_i(\langle w, x_i\rangle + b) \geq 1, \forall i \in [1, m]$ holds.

This formulation can be extended by a slack variable $\xi_i > 0$ penalizing non-zero dimensions, and a constant C, which limits the number of samples allowed to lie within the margin, e.g. when the data is not totally linear separable. As a result the optimization problem becomes minimizing w and ξ, i.e.

$$\min_{w,\xi} \frac{1}{2}||w||^2 + C \sum_{i=1}^{m} \xi_i$$

s.t.

$$y_i(\langle w, x_i\rangle + b) \geq 1 - \xi_i, \forall i \in [1, m].$$

If we want to separate non-linear separable classes, a dual problem formulation is required, additional to the primal problem just formulated. As we can represent w as linear combination of training samples, with $w = \sum_{i=1}^{m} \alpha_i y_i x_i$, we can derive the dual form via the Lagrange multipliers and the Karush-Kuhn-Tucker condition, resulting in the optimization problem to maximize α, i.e.

$$\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle$$

s.t.

$$0 \leq \alpha_i \leq C, \sum_{i=1}^{m} \alpha_i y_i = 0.$$

This results in the following classification rule:

$$f(x) = sgn(\langle w, x \rangle + b) = sgn(\sum_{i=1}^{m} \alpha_i y_i \langle x_i, x \rangle + b) \tag{7.1}$$

This also explains the origin of the name of the method, as *support vectors* are a subset of x_i, whose Lagrange variables $\alpha_i \neq 0$ and which lie either on or in the margin (if $\xi_i > 0$). We can now extend this classifier to non-linear separable classes by using kernel functions, which are based on the idea of using a high-dimensional hyperplane for separating the classes - which can be integrated in the SVM without explicitly projecting the samples into this high-dimensional feature space. This is called the 'Kernel Trick'. As such we start by defining a projection function $\phi : \mathbb{R}^{d_1} \rightarrow \mathbb{R}^{d_2}, x \mapsto \phi(x)$, with $d_1 < d_2$. Since we are using only the scalar product of x_i and x_j in the optimization problem, we can replace $\langle x_i, x_j \rangle$ in the input space $\mathbb{R}^{d_1}$ with $\langle \phi(x_i), \phi(x_j) \rangle$ in the high-dimensional feature space $\mathbb{R}^{d_2}$. Now we can use a positive definite kernel function

$$k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$$

to define the new classification method

$$f(x) = sgn(\langle w, \phi(x) \rangle + b) = sgn(\sum_{i=1}^{m} \alpha_i y_i k(x_i, x) + b) \tag{7.2}$$

with $w = \sum_{i=1}^{m} \alpha_i y_i \phi(x_i)$. An example of such a kernel function is the widely used radial basis function (RBF) kernel, defined as

$$k(x_i, x_j) = exp(-\frac{||x_i - x_j||^2}{2\sigma^2}),$$

with σ as the kernel width, which has to be set through model selection.

7.1.2 One Class SVM

Using an SVM approach for novelty or outlier detection was first discussed in [SWS$^+$00], which allowed defining a separating hyperplane for describing a single (unlabeled) data distribution in the form of a One Class SVM. Given a data distribution $x \in X$, the objective is to detect samples deviating from this distribution and label them as novel or anomalous. Similar to the previously explained Support Vector Machines, the kernel trick can be applied, i.e. one can choose a kernel to define a function $\phi(x)$, projecting the data into a high-dimensional feature space. Now a hyperplane can be defined, separating the origin and the given data distribution. By maximizing its distance from the origin, we obtain a description of the data distribution. A novel or anomalous data point is then labeled as such. Additionally an anomaly score can be assigned to this data point, e.g. based on the distance to the selected hyperplane. Figure 7.1 illustrates these concepts.

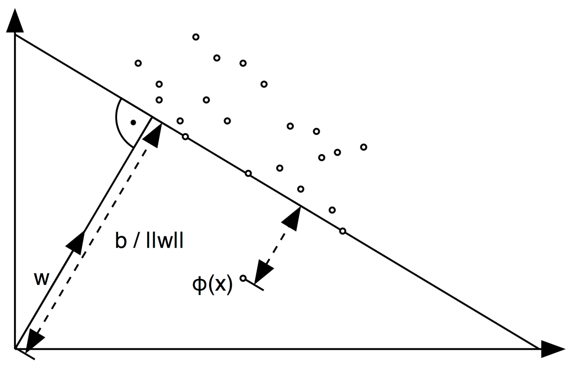


Figure 7.1: Graphical illustration of the One Class SVM

Due to its similarity to the Two Class SVM formulation, also its optimization problems are similar. Thus the primal optimization problem is again formulated as the following minimzation

$$\min_{w,b,\xi} \frac{1}{2}||w||^2 - b + \frac{1}{m\nu} \sum_{i=1}^{m} \xi_i$$

s.t.

$$\langle w, \phi(x_i) \rangle \geq b - \xi_i \text{ and } \xi_i \geq 0, \forall i \in [1, m], 0 \leq \nu \leq 1,$$

where ν allows a better model selection, providing an upper bound on the fraction of outliers. The respective dual optimization problem is formulated as follows:

$$\max_{\alpha} -\frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j k(x_i, x_j)$$

s.t.

$$\sum_{i=1}^{m} \alpha_i = 1 \text{ and } 0 \leq \alpha_i \leq \frac{1}{m\nu}, \forall i \in [1, m].$$

Similar to the SVM definition, the decision function utilizes the position of the predicted data point respective to the separating hyperplane and can be defined as

$$f(x) = sgn(\langle w, \phi(x) \rangle + b). \tag{7.3}$$

Another approach on the use of support vectors and kernels to describe a data distribution and use it for novelty detection, which is equivalent to the previously described one but slightly more intuitive, was provided in [TD04]. Here the optimization objective is formulated as minimizing an enclosing hypersphere around the data distribution, with the anomaly score of a newly classified data point being defined by its distance from the center of this hypersphere.

7.1.3 MLP

Artificial neural networks are based on the idea of reproducing the way neurons work in the brain. In this sense perceptrons [Ros58] were introduced to represent a single neuron. It receives input values of previous neurons and processes their weighted sum with an activation function, finally producing an output in the range of $[0, 1]$. Whereas the original perceptron was restricted to using thresholded activation functions though, a single neuron in a multilayer perceptron network [Sch15] allows using arbitrary activation functions. In such a *feedforward* neural network, multiple of such perceptrons are combined in multiple layers, namely the *input layer*, several *hidden layers*, and a final *output layer*. Figure 7.2 illustrates this concept, showing the different layers and the respective neuronal nodes, here using a sigmoid activation function. Notice that each previous node is fully connected to each node of the subsequent layer.

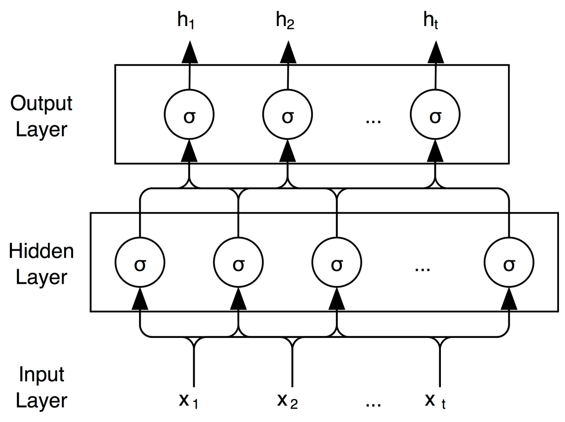


Figure 7.2: Exemplary fully-connected Multi Layer Perceptron Network

Each node has a weight vector assigned to its input data. By comparing the labels of the input data with the created output labels, an error function can be defined. During the training procedure of the MLP network the weight vectors are dynamically adapted to reduce this error function. This is achieved using a gradient descent over the calculated error surface, followed by a backpropagation of the error to the respective weight vectors. This basis of most neural network learning methods is described in detail in [RHW86].

7.1.4 RNN and LSTM

As an extension of neural network architectures to sequential data, Recurrent Neural Networks [RHW86] have been designed to enable access to information of previous events. This is achieved by extending classical feedforward neural network architectures by integrating loops into the processing of each node, illustrated in Figure 7.3. Here a node architecture A of a neural network layer processes input data x_t, i.e. data x at moment t - but additionally utilizes a loop containing the output data of the previous step $t-1$. This concept becomes clear in the unrolled node chain.

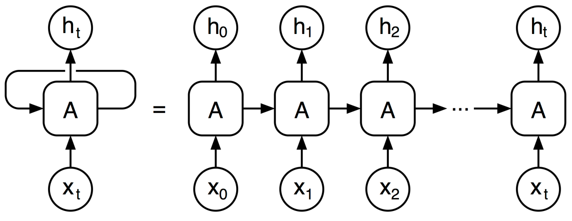


Figure 7.3: Unrolled RNN node loop

While RNNs are able to properly represent sequences with a short temporal context (i.e. contextually relevant data points are always close together), they fail in representing information represented in a larger temporal context (i.e. contextually relevant data points are further away). For this purpose Long Short Term Memory networks (LSTMs) [HS97] were designed, conceptually capable of properly representing this long-term dependency problem. Before discussing their specific properties, Figure 7.5 shows the notation used to describe those properties graphically.

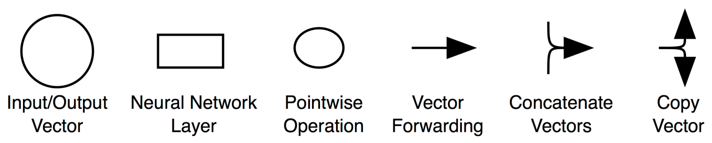


Figure 7.4: Notation for the graphical LSTM explanations

Commonly used neural network node operate using specific activation functions (also known as transfer function) to determine how strong the respective neuron is activated, by defining its output based on the summed and weighted input vectors. These activation functions are usually non-linear, to represent complex ways of combining the input values, e.g. the logistic function (also known as sigmoid function, due to its shape), producing output values in the range of $[0, 1]$, or the hyperbolic tangent function (tanh), returning values in the range of $[-1, +1]$. Figure 7.5 illustrates this with a tanh activation function.

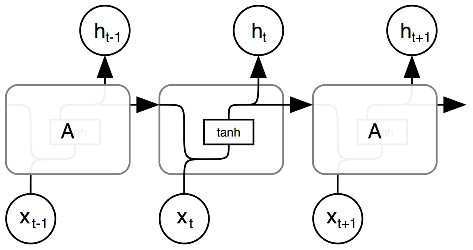


Figure 7.5: Simple activation function layout in RNN

An LSTM node consists of a complex arrangement of different activation functions. Additionally, the so called *cell state* provides a means to memorize previous events. These concepts provide an efficient way of including new relevant information and to remove older no longer relevant information. Figure 7.6 illustrates these concepts, processing the input data x_t, the previous output data h_{t-1} and the previous cell state C_{t-1}.

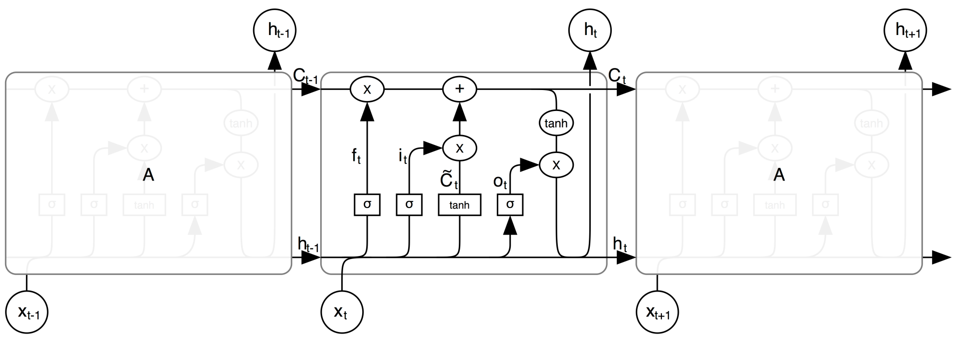


Figure 7.6: Structure of an LSTM node

The cell state is modified by so-called "gates", which consist of a sigmoid neural network layer and a point-wise operation. As its respective sigmoid layer provides a weighting in the range of $[0, 1]$, these gate allow removing or adding information as needed. The first one is the "forget gate layer", which utilizes f_t defined as follows:

$$f_t = \sigma(\langle W_f, [h_{t-1}, x_t]\rangle + b_F)$$

thereby assessing, which of the previous information to discard. The second one is the "input gate layer", creating a new vector $\tilde{C}$ to populate the cell state, which is then also filtered via the sigmoid function. This is defined as follows:

$$\tilde{C}_t = tanh(\langle W_C, [t_{t-1}, x_t]\rangle + b_C)$$

$$i_t = \sigma(\langle W_i, [h_{t-1}, x_t]\rangle + b_i)$$

Now the cell state can be updated as follows:

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

Finally the new output h_t is assembled, based on a filtered version of the previous output h_{t-1} and the updated cell state C_t.

$$o_t = \sigma(\langle W_o, [h_{t-1}, x_t]\rangle + b_o)$$

$$h_t = o_t \odot tanh(C_t)$$

7.2 Evaluation Metrics

We are using the following metrics for the evaluations conducted throughout the thesis, all of which are based on the base metrics of true positives (*TP*), false positives (*FP*), true negatives (*TN*) and false negatives (*FN*), as illustrated in Figure 7.7. Note that not all of them can be used for each type and combination of classifier, so their individual requirements are discussed separately in the respective chapters. Section 7.3 provides additional discussions on the impact of unbalanced classes on the evaluation of the system performance and the selection of the respective metrics.

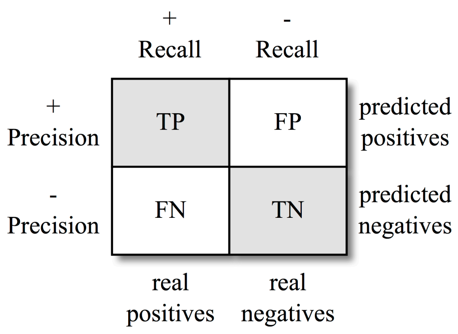


Figure 7.7: Base metrics and their interpretation

Precision

The *precision* focusses on the predicted positives. It provides a view on the ratio of correctly positive predicted values. This metric is very important, if the objective is to focus on one of the two classes, finding its - and only its - samples. Precision is also known as *true positive rate* or *confidence*, and is defined as follows:

$$\mathit{precision} = \frac{TP}{TP + FP}$$

Recall

The *recall* focusses on the real positives. It provides a view on the ratio of positively predicted samples found over all positive samples. It is also known as *sensitivity*. It is defined as follows:

$$\textit{recall} = \frac{TP}{TP + FN}$$

False Positive Rate

The *false positive rate* provides the ratio of erroneously positively predicted samples. It is defined as follows:

$$\textit{false positive rate} = \frac{FP}{TP + FP}$$

Accuracy

The *accuracy* focusses on correct predictions, i.e. it provides a view on the ratio of overall correctly predicted samples of all samples of both classes. It is defined as follows:

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

ROC-AUC

For this metric the Receiver Operation Characteristics curve is calculated over the *false positive* and *true positive* values, achieved by shifting the decision boundary of the trained classifier along the bias values defined by the decision scores (i.e. the distances) of all test samples. Afterwards its enclosed area is calculated, which has to be maximized to achieve the best classification performance. Additionally, this curve also allows for the calibration of classification models on validation data, e.g. by setting thresholds for the maximum allowed false positive rate of the final model.

$F1$ score

The F1 score is the balanced F score, which is the harmonic mean of precision and recall. It is defined as follows:

$$F1(y_{true}, y_{pred}, c) = 2 \cdot \frac{\textit{precision} \cdot \textit{recall}}{\textit{precision} + \textit{recall}}$$

The parameter c is the class label of the class that is to be predicted positively (or detected) and is thus the primary class for this F1score. It focusses on both the predicted positives and the real positives. The F1score is only defined for the binary case, i.e. for two classes, but can be extended to cover more classes by averaging procedures, as described below.

Multi-class extension

Above metrics are always defined on two classes: a positive and a negative class. Using multi-class classifiers (as e.g. in Chapter 5) we need to be able to evaluate their performance together. To achieve a better representation of potentially unweighted class sizes, we are using two ways of averaging:

- *macro* averaging, which calculates a value per positive class, and averages over all
- *weighted* averaging, which calculates a value per positive class, and weighs it with the number of true samples of this class

This is described below for the F1score, but can be calculated equivalently for accuracy, precision, recall and ROC-AUC. Thus $F1_{macro}$ is defined as follows:

$$F1score_{macro}(y_{true}, y_{pred}, C) = \frac{\sum\limits_{\forall c \in C} F1(y_{true}, y_{pred}, c)}{|C|}$$

with $|C| = |set(\forall c \in y_{true})|$, i.e. the number of unique classes c. This means, the $F1score$ is calculated for each c, and then the unweighted average is calculated. Therefore this approach does not take class size variance into account - but $F1score_{weighted}$ does, and is defined as follows:

$$F1score_{weighted}(y_{true}, y_{pred}, C) = \frac{\sum\limits_{\forall c \in C} F1(y_{true}, y_{pred}, c) \cdot |c|}{\sum\limits_{\forall c \in C} |C|}$$

with $|c|$ being the number of samples of this class c. This means, $F1$ is again calculated for each c, but the average is weighted by the number of samples of each respective c. Therefore this approach alters $F1score_{macro}$ by taking class size imbalance into account.

7.3 Using Metrics for Data with unbalanced Class Sizes

Learning on data with unbalanced class sizes requires additional care in the selection of learning methods and performance evaluation metrics. We already covered some aspects of this in Chapter 3.4.2 with the class weighting and in Chapter 5.5 with the combined classification system and the confidence ratings. By discussing the implications of the Bases-Rate Neglect and the use of the accuracy as a metric, this appendix adds further analyses on this topic. The Base-Rate Neglect describes an effect occurring in detection systems applied to sets of populations with largely differing size. In the context of IT security it was first raised in [Axe00], where it was discussed for intrusion detection systems, in which large numbers of benign reports are discriminated against very small numbers of malicious intrusion attempts - a property that is also true for the use cases discussed in Chapters 2, 3 and (to some extent) Chapter 4. The consequence of this effect is that a trained intrusion detection system requires a false positive rate (FPR) much lower than the probability that an actual intrusion $\hat{I}$ may occur ($P(\hat{I})$), i.e. $FPR << P(\hat{I})$. If this requirement is not fulfilled, it is more likely that a detected intrusion is in fact a benign sample (a false positive) than being a correctly predicted intrusion (a true positive). This is true despite otherwise high true positive rates of the applied detector.

We will now analyze this more formally. We denote I to be an incident, representing an intrusion (as in [Axe00]), a covert HTTP request (in Chapter 2), a malicious JavaScript containing URL (in Chapter 3) or an invalid failure sample (in Chapter 4). We denote A to be an alarm caused by the detection of an incident I through the respective detection system. Then we can define the following probabilities:

$P(A\|I)$	:	True Positive Rate
$P(A\|\neg I)$	:	False Positive Rate
$P(I)$	:	Probability that incident I happens
$P(\neg I)$	:	Probability that incident I does not happen

with $P(I) = \big(\text{\# of samples p.d./\# of incidents p.d.}\big)^{-1}$. These values can be placed in Bayes' Theorem

$$P(A|B) = \frac{P(A) \cdot P(B|A)}{P(B)}$$

to obtain the following formulation for calculating the Bayes Detection Rate $P(I|A)$:

$$P(I|A) = \frac{P(I) \cdot P(A|I)}{P(I) \cdot P(A|I) + P(\neg I) \cdot P(A|\neg I)}$$

By providing the probability that an alarm A is indeed correctly predicting an incident I, $P(I|A)$ allows an intuitive interpretation of the practical usability of detection systems for unbalanced class data. We will now shortly discuss this for some examples. For these examples also the respective Accuracy values are shown. As the accuracy balances true positive and true negative values equivalently, it is highly prone to reflect the detection performance on the benign class, achieving indifferentiable results in the process. Due to this limited capability of the accuracy to properly reflect the classification performance of system applied to data with very unbalanced class sizes, it was not used in this thesis.

Table 7.1 shows the relevant properties of exemplarily selected data sets of Chapters 2, 3 and 4. Their Bayes Detection Rates $P(A|I)$ and their accuracy values are shown in Table 7.2. For the DUMONT detector the resulting probability that an alarm is indeed caused by a covert communication attempt is very high,

	Chapter 2 Requests		Chapter 3 Filtered URLs		Chapter 4 2014 Campaign	
	Benign	Malicious	Benign	Malicious	Valid	Invalid
Timespan	90 days		137 days		22 days	
Samples	182,996	12,899	2,900,000	3,220	4,500	400
Samples p.d.	2,033	143	21,267	23	205	18
$P(I)$	0.066		0.001		0.081	
$P(\neg I)$	0.934		0.999		0.919	

Table 7.1: Representative data set properties of Chapters 2, 3 and 4

	Chapter 2	Chapter 3		Chapter 4
	DUMONT Detector	Competing AV Scanners	Combined Detectors	Exemplary SVM($\Gamma_{s.less}$)
Accuracy	0.9873	0.9987	0.9994	0.9147
$P(A\|I)$	0.857	0.295	0.545	0.901
$P(A\|\neg I)$	0.0035	0.0005	0.0001	0.084
$P(I\|A)$	0.95	0.371	0.845	0.485

Table 7.2: Representative classification model properties of Chapters 2, 3 and 4

i.e. $P(I|A) = 95\%$, making these predictions very reliable. In Chapter 3 the probability that an alarm is indeed caused by a JavaScript attack is 37.1 % for the competing AV Scanners and 84.5 % for the proposed combined detectors (on all samples). Thus the Bayes Detection Rate allows an easy identification of the better model - while the accuracy barely changes for the two models at all and does therefore not allow a proper performance comparison. The selected SVM classifier on $\Gamma_{s.less}$ features, representing the classification performance of the system proposed in Chapter 4, achieves a $P(I|A)$ of 48.5%. Hence each correctly predicted invalid sample also adds (approximately) one incorrectly predicted valid sample. Nonetheless the predictor remains useable, since the ratio of valid to invalid samples is merely $11.25 : 1$, barely meeting the assumption of the Base-Rate Neglect. Hence this approach still provides a filter, reliably preselecting candidate samples. For the analysis of these candidates the proposed methods for reliability estimation, like the the weight vector visualization of Chapter 4.6.2 and the confidence ratings of Chapter 5.5 provide additional help. This percentage can also be further alleviated by choosing other models with a lower $P(A|\neg I)$ or applying the class weighting, described in Chapter 3.4.2. Since the class data used for the evaluations of Chapter 5 are relatively balanced, the assumption of the Base Rate Neglect does not hold. Therefore we will not discuss its Bayes Detection Rate here.

7.4 Visualizations for Interpretable Systems

To be able to effectively utilize machine learning approaches in productive systems, they need to address more aspects than just the classification performance. To achieve a trustworthy system, transparence of the learning procedure and the utilized features, as well as interpretability of the classification results are equally important. In this section we will discuss two visualization methods specifically developed to address those topics. As such Section 7.4.1 provides an extended discussion of the weight vector visualization introduced in Section 4.6.2, which allows additional insights into the classification results on whole data sets. Section 7.4.2 on the other hand discusses a way of visualizing the structural $\delta - n$ matching function $\hat{\Phi}$, as defined for the Γ_{ST} feature space in Section 5.4.6, providing insights into this feature space projection procedure. Further analyses on the relevance of the interpretability of more general classification systems for reliable results can be found in [LWB$^+$19].

7.4.1 Weight Vector Visualization

As already discussed in Section 4.6.2, we can visualize the values of the individual dimensions of the w of a trained SVM model to estimate the relevance of the respective dimensions for the underlying classification model. Table 7.3 illustrates such exemplary dimensions and its values w_i in the weight vector w, on the example of the token 3-gram debug-data of Chapter 4. As discussed there, we can also further improve the interpretability of these values by decorrelating w using the covariance matrix of S_{train}. To highlight its impact, figures 7.8, 7.9, 7.10 and 7.11 show the visualizations of the original correlated and the decorrelated w of a whole data set, for models trained with the L1 and L2 norm.

w_i	token 3-gram dimension d_i
-0.674	Redirect to https
-0.431	028 TASK_RESULT HTTP
$\dots$	...
-0.124	175 SP_ICMP EV_SA_SUCCESS
$\dots$	...
-0.001	028 STATEMACHINE VMCCRAT
$\dots$	...
0.075	175 DTS MODULE
$\dots$	...
0.286	23 kbit/s smart
0.323	Overall Duration 10000

Table 7.3: Exemplary token 3-gram dimensions of w of debug-data

As described in Chapter 4, the bottom row shows the respective values of w, sorted descending. Each row contains the data for a single sample x. The first parameter denotes the sample ID, followed by the y_{true} and y_{pred}, the true and the predicted class label. This is followed by an *identity verification* of $y_{true} = y_{pred}$, which is denoted by the letter "T", if found *true*. Finally the $\sum w_i \forall d_i \in x$ is shown, i.e. the sum of the w_i of those dimensions d_i that occur in the sample x - all of which are then shown as the black bars in the graphical representations.

By creating these visualizations for a whole data set, sorted by the sum of the matching weights, we can determine dimensions that are relevant for whole subsets of samples, indicating a potential class relationship between those samples. The visualization also allows conclusions about the confidence in the predictions for the individual samples: whereas the samples at the top and the bottom end are predominantly classified correctly (lying farthest from the separating hyperplane of the utilized SVM), those in the middle lie closer to the separating hyperplane, increasing the risk for potential misclassifications. This can easily be seen when closer inspecting the *identity verification*, which shows most misclassifications around this region.

When using the L1 norm during the training phase, the dimensions are already distinguishably separated, s.t. the interpretation of the individual dimensional relevance is in fact easier on the original sparse solution, than on the dense, decorrelated version, where many additional dimensions suddenly become equally relevant. However, in our experiments the L2 norm generally allowed a better classification performance, and is therefore the preferred approach. In this case, as shown in figures 7.10 and 7.11, a comprehensive interpretation of the relevant dimensions absolutely requires the decorrelation of w, as its weights w_i lie otherwise indistinguishably close to each other.

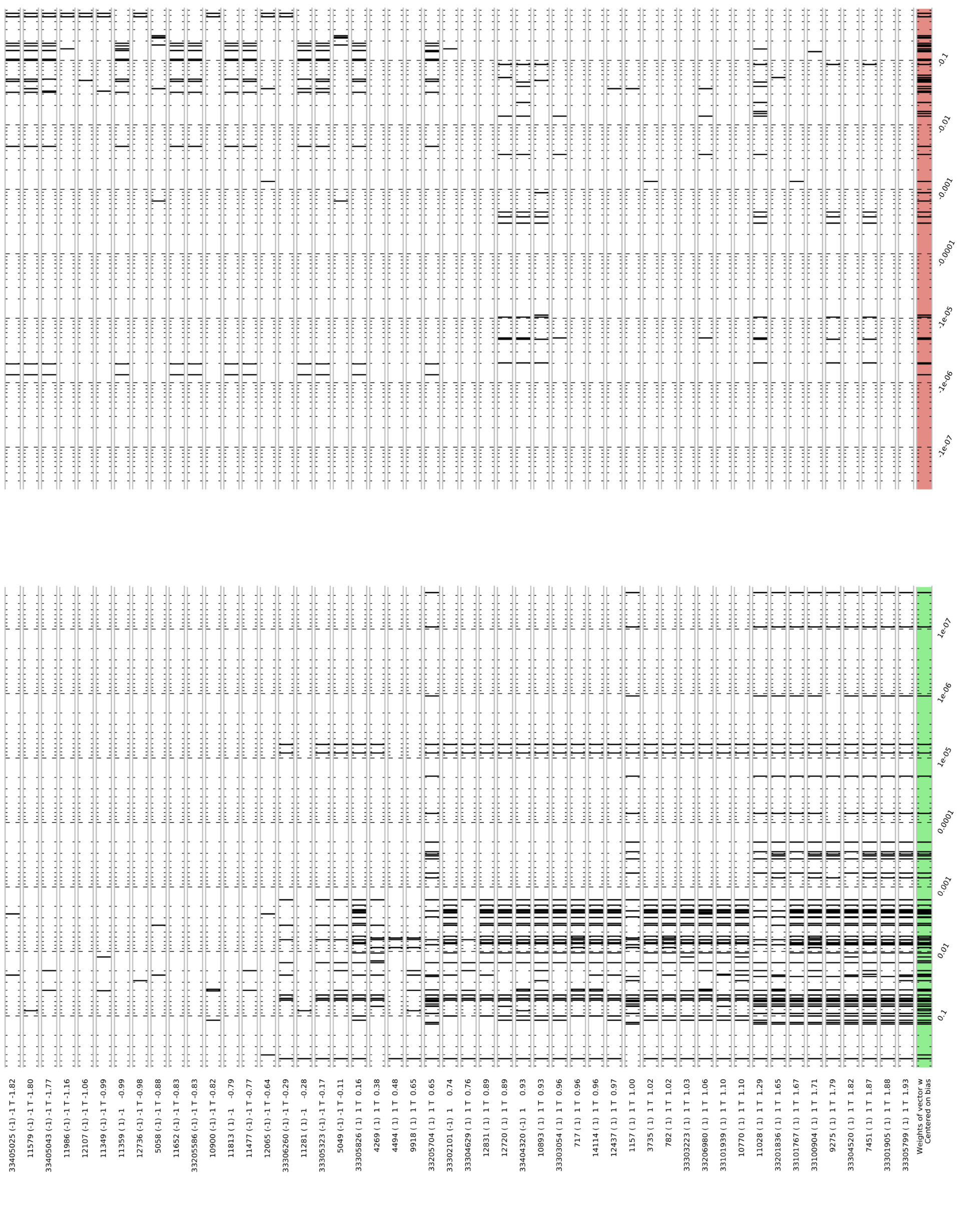


Figure 7.8: w-visualizations based on the L1 norm, correlated with S_{train}

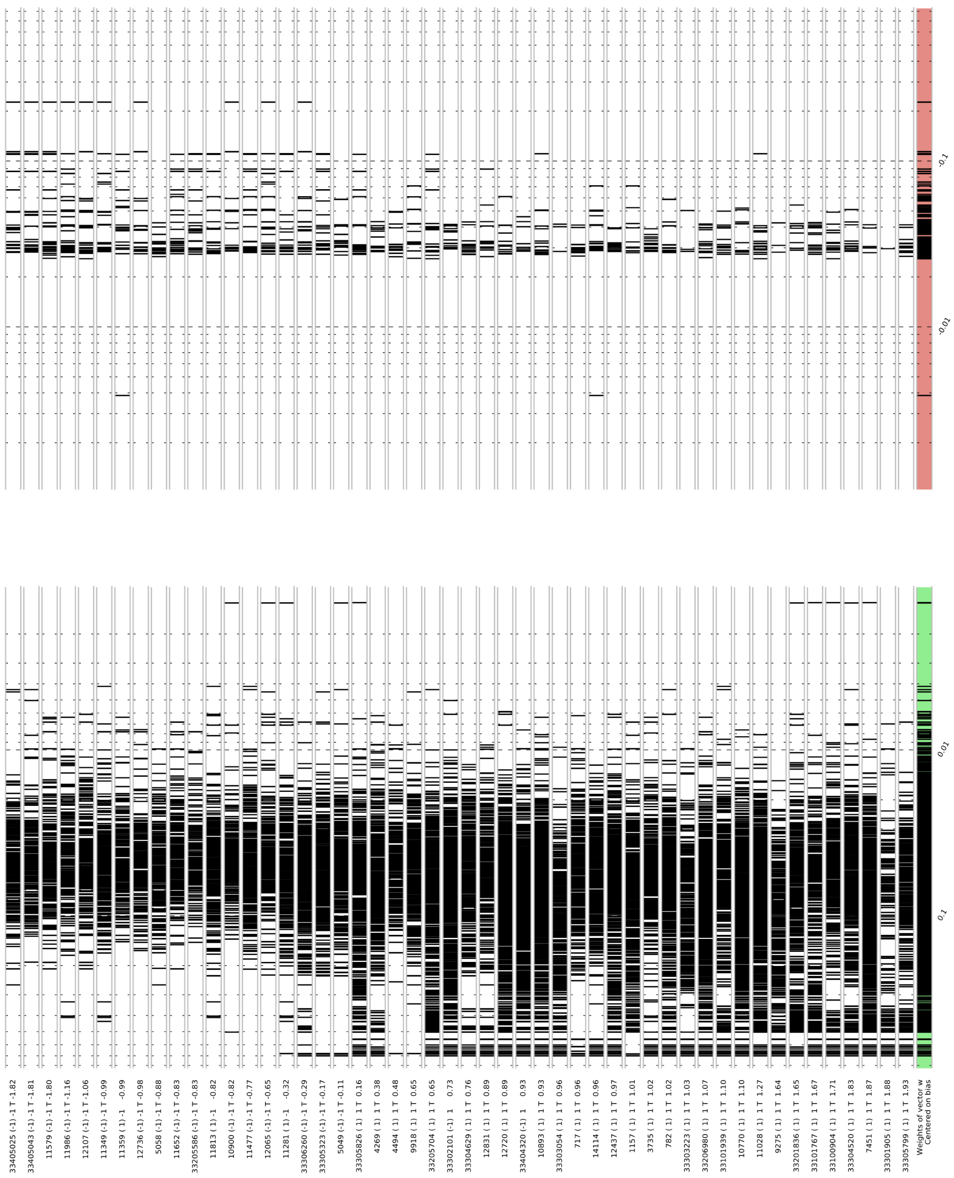


Figure 7.9: w-visualizations based on the L1 norm, decorrelated from S_{train}

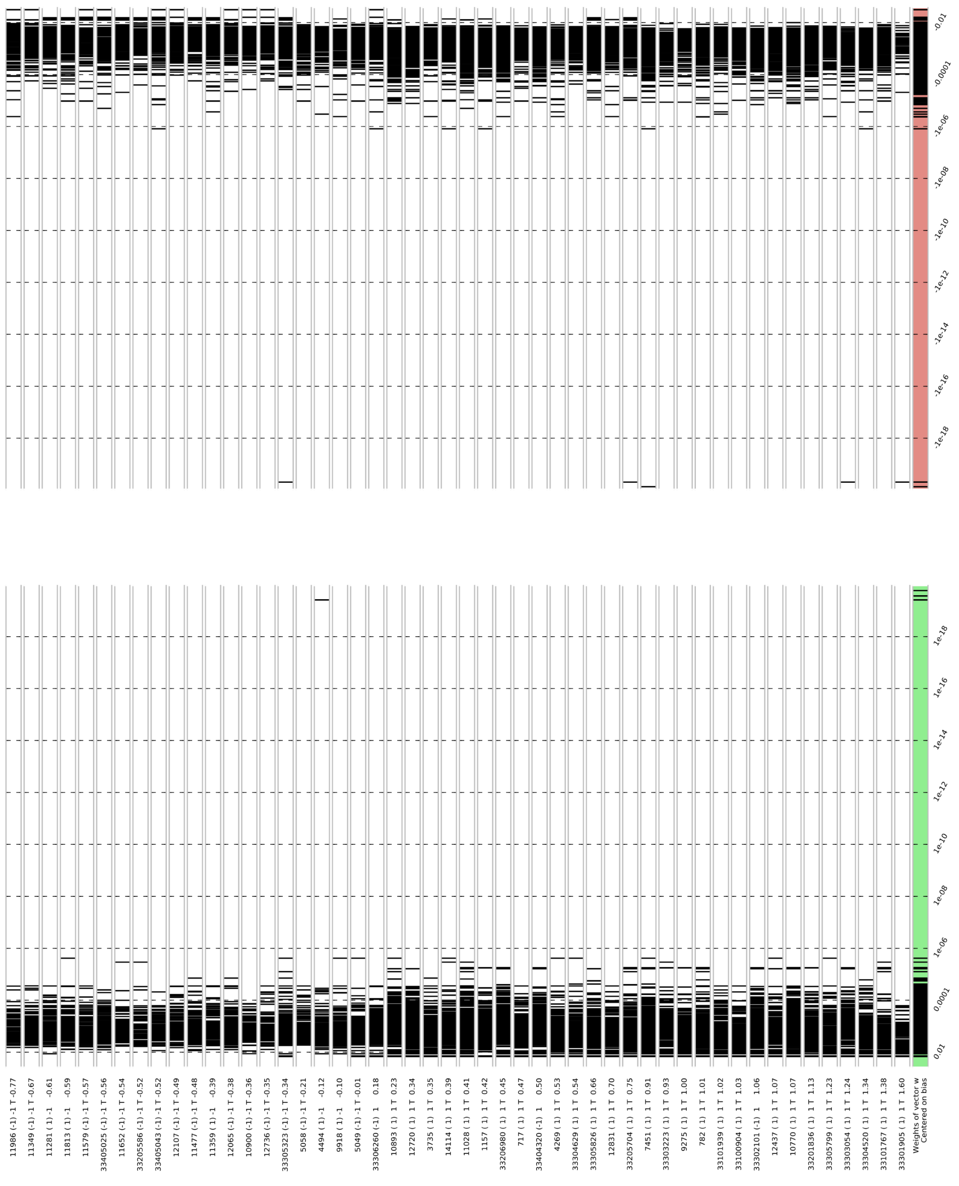


Figure 7.10: w-visualizations based on the L2 norm, correlated with S_{train}

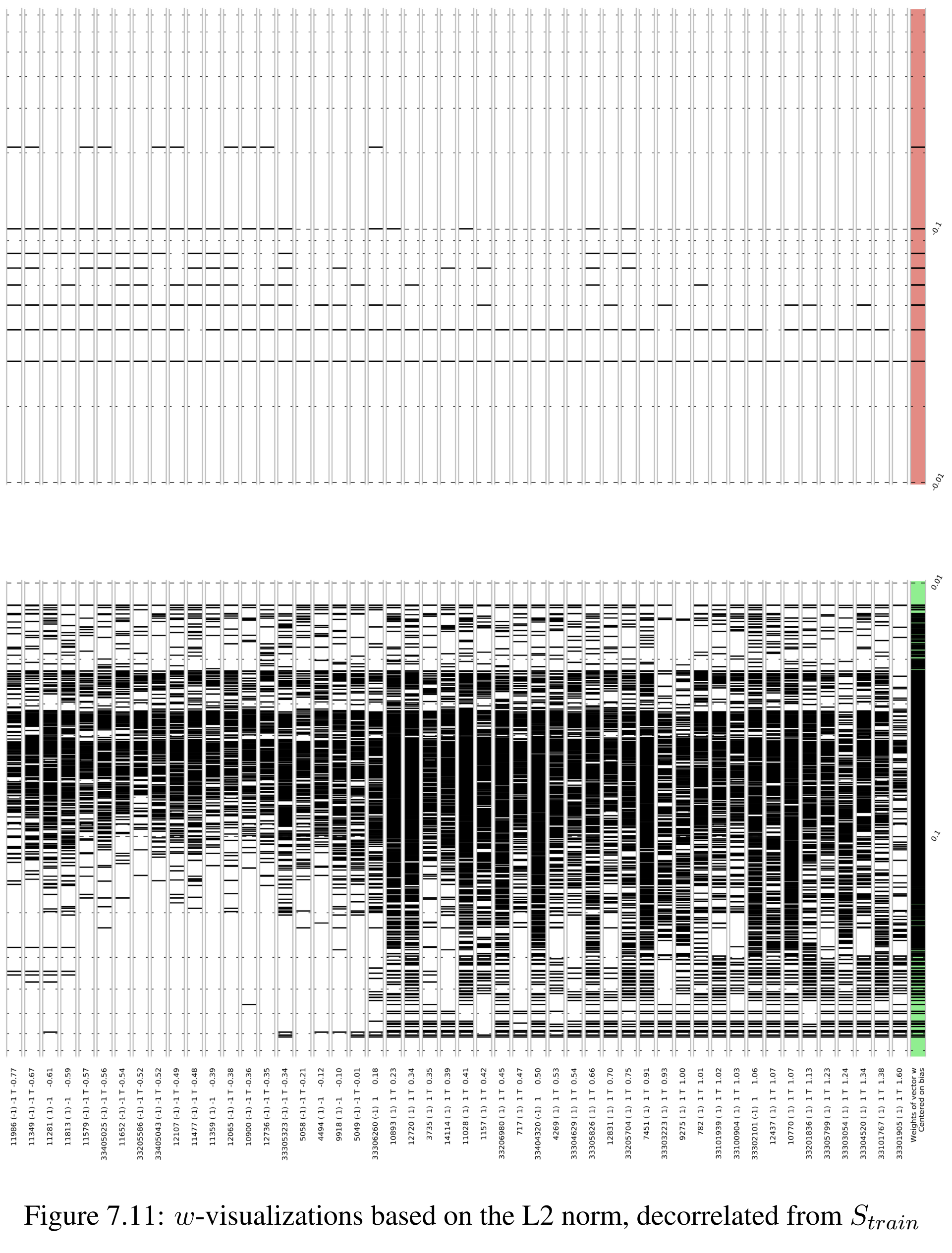


Figure 7.11: w-visualizations based on the L2 norm, decorrelated from S_{train}

7.4.2 Visualizing the $\delta - n$ matching function $\hat{\Phi}$ of Γ_{ST}

To obtain additional insights into the $\delta - n$ matching procedure, as defined for the Γ_{ST} feature space in Section 4.6.2, we developed a tree-based visualization, as shown exemplarily in Figure 7.12. The idea is to obtain the ability to visually inspect similarities between a projected sequence s and a selected set of model sequences S^M, utilized in the structural $\delta - n$ matching function $\hat{\Phi}$. To achieve this, the model sequences $s^M \in S^M$ with $S^M \in \{S^M_{2C}, S^M_{MOC}, S^M_{MTC}\}$ are modeled as a tree, with the events as individual nodes. To reduce the number of depicted nodes, the tree is converted into a prefix tree, i.e. the prefix-paths of sequences are merged, if they are identical. To further reduce the complexity, also the suffix-paths can be merged, resulting in a compressed graph, depicted in Figure 7.13, which also contain the final colored highlights for the structural $\delta - n$ matching, for values of $n = 2$ and $\delta = 1$. To allow the visual inspection, nodes of the projected sequence s are highlighted with a *bold black* frame, and their structural matches with *bold blue* frames. Using these colored highlights, the types of sequences contained in the visualized S^M, as well as relevant or dominant sub-sequences can be easily distinguished.

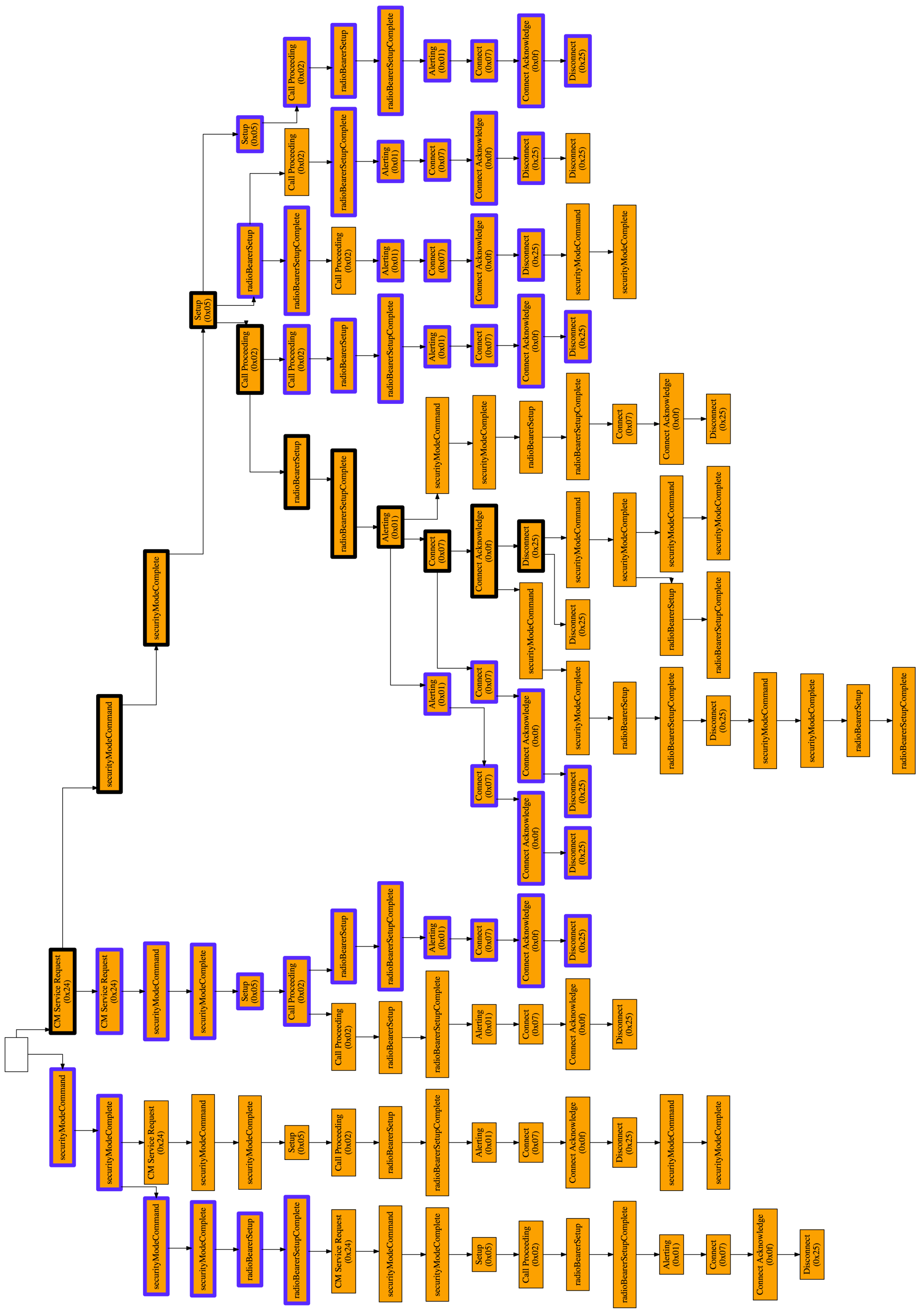


Figure 7.12: Prefix-tree visualization of sequence s projected on model sequences S^M

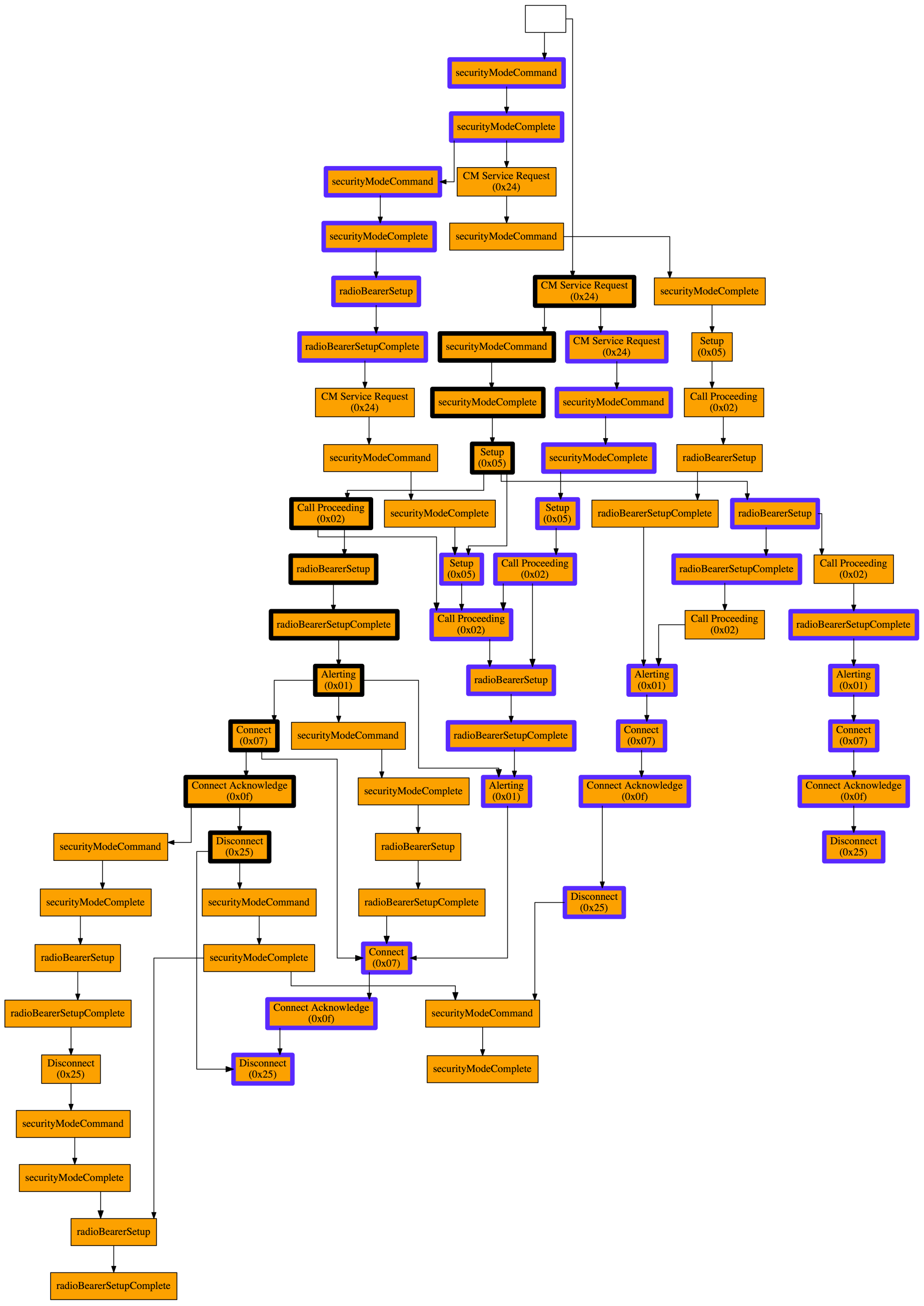


Figure 7.13: Compressed graph visualization of sequence s projected on model sequences S^M

Bibliography

[AAB$^+$15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[ABCM09] Michal Aharon, Gilad Barash, Ira Cohen, and Eli Mordechai. *One Graph Is Worth a Thousand Logs: Uncovering Hidden Structures in Massive System Event Logs*, pages 227–243. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[Axe00] Stefan Axelsson. The base-rate fallacy and the difficulty of intrusion detection. *ACM Transactions on Information and System Security (TISSEC)*, 3(3):186–205, 2000.

[BBCP04] J. Beale, A.R. Baker, B. Caswell, and M. Poor. *Snort 2.1 Intrusion Detection*. Syngress Publishing, 2nd edition, 2004.

[BBM08] Mikio L Braun, Joachim M Buhmann, and Klaus Robert Müller. On relevant dimensions in kernel feature spaces. *Journal of Machine Learning Research*, 9:1875–1908, 2008.

[BBM$^+$15] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLOS ONE*, 10(7):1–46, 07 2015.

[BGH14] Georges Bossert, Frédéric Guihéry, and Guillaume Hiet. Towards automated protocol reverse engineering using semantic information. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pages 51–62. ACM, 2014.

[Bis95] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., New York, NY, USA, 1995.

[Blo70] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[BMB10] Armin Büscher, Michael Meier, and Ralf Benzmüller. Throwing a monkeywrench into web attackers plans. In *Proc. of Communications and Multimedia Security (CMS)*, 2010.

[BP04] K. Borders and A. Prakash. Web tap: detecting covert web traffic. In *Proc. of Conference on Computer and Communications Security (CCS)*, pages 110–120, 2004.

[BPX$^+$07] Thorsten Brants, Ashok C Popat, Peng Xu, Franz J Och, and Jeffrey Dean. Large language models in machine translation. In *In Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 858–867. Citeseer, 2007.

[BS85] R. Beran and M.S. Srivastava. Bootstrap tests and confidence regions for functions of a covariance matrix. *Annals of Statistics*, 13(1):95–115, 1985.

[BSH$^+$10] David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Müller. How to explain individual classification decisions. *J. Mach. Learn. Res.*, 11:1803–1831, August 2010.

[CCVK11] D. Canali, M. Cova, G. Vigna, and C. Kruegel. Prophiler: a fast filter for the large-scale detection of malicious web pages. In *Proc. of the International World Wide Web Conference (WWW)*, April 2011.

[CFH14] Woon Hau Chin, Zhong Fan, and Russell Haines. Emerging technologies and research challenges for 5g wireless networks. *IEEE Wireless Communications*, 21(2):106–112, 2014.

[CKV10] Marco Cova, Christopher Kruegel, and Giovanni Vigna. Detection and analysis of drive-by-download attacks and malicious

JavaScript code. In *Proc. of the International World Wide Web Conference (WWW)*, 2010.

[CLC$^+$16] Patrik Cerwall, Anette Lundvall, Stephen Carson, Richard Möller, Susanna Bävertoft, Anna Jacobsson, Git Sellin, Michael Björn, Vishnu Singh, Stephen Carson, Reiner Ludwig, Lasse Wieweg, Jonas Edstam, Per Lindberg, and Kati Öhman. Ericsson mobility report: On the pulse of the networked society. Technical report, Ericsson, Stockholm, Sweden, June 2016.

[CLF$^+$14] Michelangelo Ceci, Pasqua Fabiana Lanotte, Fabio Fumarola, Dario Pietro Cavallo, and Donato Malerba. Completion time and next activity prediction of processes using sequential pattern mining. In *International Conference on Discovery Science*, pages 49–61. Springer, 2014.

[CLZS11] Charlie Curtsinger, Benjamin Livshits, Benjamin Zorn, and Christian Seifert. Zozzle: Fast and precise in-browser javascript malware detection. In *Proc. of USENIX Security Symposium*, 2011.

[Col02] Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 1–8. Association for Computational Linguistics, 2002.

[CPC$^+$08] Weidong Cui, Marcus Peinado, Karl Chen, Helen J Wang, and Luis Irun-Briz. Tupni: Automatic reverse engineering of input formats. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 391–402. ACM, 2008.

[CPWK06] Weidong Cui, Vern Paxson, Nicholas Weaver, and Randy H Katz. Protocol-independent adaptive replay of application dialog. In *NDSS*, volume 4, pages 279–293, 2006.

[CSL$^+$08] G. Cretu, A. Stavrou, M. Locasto, S.J. Stolfo, and A.D. Keromytis. Casting out demons: Sanitizing training data for anomaly sensors. In *Proc. of IEEE Symposium on Security and Privacy*, 2008.

[CSRL01] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[CSS+10] Chia Yuan Cho, Eui Chul Richard Shin, Dawn Song, et al. Inference and analysis of formal models of botnet command and control protocols. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 426–439. ACM, 2010.

[CV95] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[CWKK09] Paolo Milani Comparetti, Gilbert Wondracek, Christopher Kruegel, and Engin Kirda. Prospex: Protocol specification extraction. In *Security and Privacy, 2009 30th IEEE Symposium on*, pages 110–125. IEEE, 2009.

[CYLS07] Juan Caballero, Heng Yin, Zhenkai Liang, and Dawn Song. Polyglot: Automatic extraction of protocol message format using dynamic binary analysis. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 317–329. ACM, 2007.

[DHF10] Andreas Dewald, Thorsten Holz, and Felix C Freiling. Adsandbox: Sandboxing javascript to fight malicious websites. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 1859–1864. ACM, 2010.

[DHM08] Mark Daniel, Jake Honoroff, and Charlie Miller. Engineering heap overflow exploits with JavaScript. In *Proc. of USENIX Workshop on Offensive Technologies (WOOT)*, 2008.

[Die00] Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[Die02] Thomas G Dietterich. Machine learning for sequential data: A review. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 15–30. Springer, 2002.

[DL15] Andrew M Dai and Quoc V Le. Semi-supervised sequence learning. In *Advances in neural information processing systems*, pages 3079–3087, 2015.

[EFG+10] Markus Engelberth, Felix Freiling, Jan Goebel, Christian Gorecki, Thorsten Holz, Ralf Hund, Philipp Trinius, and Carsten Willems. The InMAS approach. In *Proc. of European Workshop on Internet Early Warning and Network Intelligence (EWNI)*, January 2010.

[EKK09] Manuel Egele, Engin Kirda, and Christopher Kruegel. Mitigating drive-by download attacks: Challenges and open problems. In *Proc. of Open Research Problems in Network Security Workshop (iNetSec)*, 2009.

[ERF16] Joerg Evermann, Jana-Rebecca Rehse, and Peter Fettke. A deep learning approach for predicting process behaviour at runtime. In *International Conference on Business Process Management*, pages 327–338. Springer, 2016.

[EWKK09] Manuel Egele, Peter Wurzinger, Christopher Kruegel, and Engin Kirda. Defending browsers against drive-by downloads: Mitigating heap-spraying code injection attacks. In *Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, 2009.

[FBH+02] N. Feamster, M. Balazinska, G. Harfst, H. Balakrishnan, and D. R. Karger. Infranet: Circumventing web censorship and surveillance. In *Proc. of USENIX Security Symposium*, pages 247–262, 2002.

[FCH+08] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research (JMLR)*, 9:1871–1874, 2008.

[FGM+99] 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. Updated by RFC 2817.

[FPPS07] Jason Franklin, Vern Paxson, Adrian Perrig, and Stefan Savage. An Inquiry Into the Nature and Causes of the Wealth of Internet Miscreants. In *Proc. of Conference on Computer and Communications Security (CCS)*, pages 375–388, 2007.

[GDDC97] David Grove, Greg DeFouw, Jeffrey Dean, and Craig Chambers. Call graph construction in object-oriented languages. In *Proceedings of the 12th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA '97, pages 108–124, New York, NY, USA, 1997. ACM.

[Gor08] Christian Gorecki. Truman box: Safe malware analysis by simulating the internet. Master's thesis, University of Mannheim, 2008.

[GPZL08] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection. In *Proc. of USENIX Security Symposium*, 2008.

[Gra12] Alex Graves. Supervised sequence labelling with recurrent neural networks. *ISBN 9783642212703. URL http://books. google. com/books*, 2012.

[GSZ13] M Ghiassi, J Skinner, and D Zimbra. Twitter brand sentiment analysis: A hybrid system using n-gram analysis and dynamic artificial neural network. *Expert Systems with applications*, 40(16):6266–6282, 2013.

[GWY$^+$15] Hugo Gascon, Christian Wressnegger, Fabian Yamaguchi, Daniel Arp, and Konrad Rieck. Pulsar: Stateful black-box fuzzing of proprietary network protocols. In *Security and Privacy in Communication Networks*, pages 330–347. Springer, 2015.

[GYAR13] Hugo Gascon, Fabian Yamaguchi, Daniel Arp, and Konrad Rieck. Structural detection of android malware using embedded call graphs. In *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*, pages 45–54. ACM, 2013.

[GZL08] G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. In *Proc. of Network and Distributed System Security Symposium (NDSS)*, 2008.

[HEF09] Thorsten Holz, Markus Engelberth, and Felix Freiling. Learning More About the Underground Economy: A Case-Study of Keyloggers and Dropzones. In *Proc. of European Symposium on Research in Computer Security (ESORICS)*, 2009.

[HFH11] Mario Heiderich, Tilman Frosch, and Thorsten Holz. IceShield: Detection and mitigiation of malicious websites with a frozen dom. In *Recent Adances in Intrusion Detection (RAID)*, September 2011.

[HGX$^+$17] Zhen He, Shaobing Gao, Liang Xiao, Daxue Liu, Hangen He, and David Barber. Wider and deeper, cheaper and faster: Tensorized lstms for sequence learning. In *Advances in Neural Information Processing Systems*, pages 1–11, 2017.

[HMG$^+$14] Stefan Haufe, Frank Meinecke, Kai Görgen, Sven Dähne, John-Dylan Haynes, Benjamin Blankertz, and Felix Bießmann. On the interpretation of weight vectors of linear models in multivariate neuroimaging. *Neuroimage*, 87:96–110, 2014.

[Hop71] John E. Hopcroft. An n log n algorithm for minimizing states in a finite automaton. Technical report, Stanford, CA, USA, 1971.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[HSD+08] T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. Freiling. Measurements and mitigation of peer-to-peer-based botnets: A case study on storm worm. In *Proc. of USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2008.

[IBC+12] Luca Invernizzi, Stefano Benvenuti, Paolo Milani Comparetti, Marco Cova, Christopher Kruegel, and Giovanni Vigna. EVILSEED: A guided approach to finding malicious web pages. In *Proc. of IEEE Symposium on Security and Privacy*, May 2012.

[IHF08] Ali Ikinci, Thorsten Holz, and Felix Freiling. Monkey-spider: Detecting malicious websites with low-interaction honeyclients. In *Proc. of Conference "Sicherheit, Schutz und Zuverlässigkeit" (SICHERHEIT)*, pages 891–898, 2008.

[II07] K. L. Ingham and H. Inoue. Comparing anomaly detection techniques for HTTP. In *Recent Adances in Intrusion Detection (RAID)*, pages 42 – 62, 2007.

[KGKR12] Tammo Krueger, Hugo Gascon, Nicole Krämer, and Konrad Rieck. Learning stateful models for network honeypots. In *Proceedings of the 5th ACM workshop on Security and artificial intelligence*, pages 37–48. ACM, 2012.

[KKR11] Tammo Krueger, Nicole Krämer, and Konrad Rieck. *ASAP: Automatic Semantics-Aware Analysis of Network Payloads*, pages 50–63. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[KL10] Marius Kloft and Pavel Laskov. Online anomaly detection under adversarial impact. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 405–412, 2010.

[KL14] Eric D Knapp and Joel Thomas Langill. *Industrial Network Security: Securing critical infrastructure networks for smart grid, SCADA, and other Industrial Control Systems*. Syngress, 2014.

[KLN+11] Sandeep Karanth, Srivatsan Laxman, Prasad Naldurg, Ramarathnam Venkatesan, J. Lambert, and Jinwook Shin. ZDVUE: Prioritization of javascript attacks to surface new vulnerabilities.

In *Proc. of CCS Workshop on Security and Artificial Intelligence (AISEC)*, October 2011.

[KLZS12] Clemens Kolbitsch, Benjamin Livshits, Benjamin Zorn, and Christian Seifert. Rozzle: de-cloaking internet malware. In *Proc. of IEEE Symposium on Security and Privacy*, May 2012.

[KPPK10] V. Kemerlis, V. Pappas, G. Portokalidis, and A. Keromytis. ileak: A lightweight system for detecting inadvertent information leaks. In *Proc. of European Conference on Computer Network Defense (EC2ND)*, 2010.

[KR05] Eamonn Keogh and Chotirat Ann Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and information systems*, 7(3):358–386, 2005.

[KSA$^+$17] Pieter-Jan Kindermans, Kristof T Schütt, Maximilian Alber, Klaus-Robert Müller, and Sven Dähne. Patternnet and patternlrp - improving the interpretability of neural networks. *arXiv:1705.05598*, 2017.

[LC11] C. Leita and M. Cova. HARMUR: Storing and analyzing historic data on malicious domains. In *Proc. of Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*, April 2011.

[LCDF$^+$15] Anna Leontjeva, Raffaele Conforti, Chiara Di Francescomarino, Marlon Dumas, and Fabrizio Maria Maggi. Complex symbolic sequence encodings for predictive monitoring of business processes. In *International Conference on Business Process Management*, pages 297–313. Springer, 2015.

[LGKM06] P. Laskov, C. Gehl, S. Krüger, and K. R. Müller. Incremental support vector learning: Analysis, implementation and applications. *Journal of Machine Learning Research*, 7:1909–1936, September 2006.

[LGN12] Mai Le, Bogdan Gabrys, and Detlef Nauck. A hybrid model for business process event prediction. In *Research and Development in Intelligent Systems XXIX*, pages 179–192. Springer, 2012.

[LJXZ08] Zhiqiang Lin, Xuxian Jiang, Dongyan Xu, and Xiangyu Zhang. Automatic protocol format reverse engineering through context-aware monitored execution. In *NDSS*, volume 8, pages 1–15, 2008.

[LMD05] Corrado Leita, Ken Mermoud, and Marc Dacier. Scriptgen: an automated script generation tool for honeyd. In *Computer Security Applications Conference, 21st Annual*. IEEE, 2005.

[LMP01] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *ICML '01 Proceedings of the Eighteenth International Conference on Machine Learning 8*, pages 282–289, 2001.

[LS05] Ming Li and Ronan Sleep. A robust approach to sequence classification. In *Tools with Artificial Intelligence, 2005. ICTAI 05. 17th IEEE International Conference on*, pages 5–pp. IEEE, 2005.

[LŠ11] Pavel Laskov and Nedim Šrndić. Static detection of malicious javascript-bearing pdf documents. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 373–382. ACM, 2011.

[LSD$^+$15] Geetika T Lakshmanan, Davood Shamsi, Yurdaer N Doganata, Merve Unuvar, and Rania Khalaf. A markov prediction model for data-driven semi-structured business processes. *Knowledge and Information Systems*, 42(1):97–126, 2015.

[LSTCW01] Huma Lodhi, John Shawe-Taylor, Nello Cristianini, and Christopher JCH Watkins. Text classification using string kernels. In *Advances in neural information processing systems*, pages 563–569, 2001.

[LWB$^+$19] Sebastian Lapuschkin, Stephan Wäldchen, Alexander Binder, Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Unmasking clever hans predictors and assessing what machines really learn. *Nature communications*, 10(1):1096, 2019.

[LWR$^+$10] Philipp Leitner, Branimir Wetzstein, Florian Rosenberg, Anton Michlmayr, Schahram Dustdar, and Frank Leymann. Runtime prediction of service level agreement violations for composite services. In *Service-oriented computing. ICSOC/ServiceWave 2009 workshops*, pages 176–186. Springer, 2010.

[LXLZ15] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. In *AAAI*, pages 2267–2273, 2015.

[LYPL10] L. Lu, V. Yegneswaran, P. A. Porras, and W. Lee. BLADE: An attack-agnostic approach for preventing drive-by malware infections. In *Proc. of Conference on Computer and Communications Security (CCS)*, October 2010.

[MBKM13] G. Montavon, M. L. Braun, T. Krueger, and K. R. Muller. Analyzing local structure in kernel-based learning: Explanation, complexity, and reliability assessment. *IEEE Signal Processing Magazine*, 30(4):62–74, July 2013.

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[McH95] John McHugh. *Handbook for the Computer Security Certification of Trusted Systems*, chapter Covert Channel Analysis. Naval Research Laboratory, 1995.

[MDFDG14] Fabrizio Maria Maggi, Chiara Di Francescomarino, Marlon Dumas, and Chiara Ghidini. Predictive monitoring of business processes. In *International Conference on Advanced Information Systems Engineering*, pages 457–472. Springer, 2014.

[Min67] Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.

[MLB$^+$17] Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognition*, 65:211 – 222, 2017.

[MLI$^+$15] Andreas Metzger, Philipp Leitner, Dragan Ivanović, Eric Schmieders, Rod Franklin, Manuel Carro, Schahram Dustdar, and Klaus Pohl. Comparing and combining predictive business process monitoring techniques. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(2):276–290, 2015.

[MMR$^+$01] K-R Muller, Sebastian Mika, Gunnar Ratsch, Koji Tsuda, and Bernhard Scholkopf. An introduction to kernel-based learning algorithms. *IEEE transactions on neural networks*, 12(2):181–201, 2001.

[MOM12] Grégoire Montavon, Genevieve B Orr, and Klaus-Robert Müller. *Neural networks: tricks of the trade*. Springer, 2012.

[MSM18] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73:1–15, 2018.

[Naz09] J. Nazario. A virtual client honeypot. In *Proc. of USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2009.

[NBFS06] James Newsome, David Brumley, Jason Franklin, and Dawn Song. Replayer: Automatic protocol replay by binary analysis. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 311–321. ACM, 2006.

[NKS05] J. Newsome, B. Karp, and D. Song. Polygraph: Automatically generating signatures for polymorphic worms. In *Proc. of IEEE Symposium on Security and Privacy*, pages 120–132, 2005.

[ORLS14] Aditya Oza, Kevin Ross, Richard M Low, and Mark Stamp. Http attack detection using n-gram analysis. *Computers & Security*, 45:242–254, 2014.

[PAF$^+$09] Roberto Perdisci, Davide Ariu, Prahlad Fogla, Giorgio Giacinto, and Wenke Lee. Mcpad: A multiple classifier system for accurate payload-based anomaly detection. *Computer networks*, 53(6):864–881, 2009.

[Pax99] V. Paxson. Bro: A system for detecting network intruders in real-time. *Computer Networks*, 31(23–24):2435–2466, December 1999.

[PMRM08] Niels Provos, Panayiotis Mavrommatis, Moheeb A. Rajab, and Fabian Monrose. All Your iFRAMEs Point to Us. In *Proc. of USENIX Security Symposium*, 2008.

[PNC11] Suraj Pandey, Surya Nepal, and Shiping Chen. A test-bed for the evaluation of business process prediction techniques. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2011 7th International Conference on*, pages 382–391. IEEE, 2011.

[PSBDL18] Mirko Polato, Alessandro Sperduti, Andrea Burattin, and Massimiliano De Leoni. Time and activity sequence prediction of business process instances. *Computing*, pages 1–27, 2018.

[PSC15] Joris Pelemans, Noam Shazeer, and Ciprian Chelba. Pruning sparse non-negative matrix n-gram language models. In *Proceedings of Interspeech*, pages 1433–1437, 2015.

[PVG$^+$11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Rab89] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[RHW86] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[Rie09] Konrad Rieck. Machine learning for application-layer intrusion detection. *Technology*, page 151, 2009.

[RKD10] Konrad Rieck, Tammo Krueger, and Andreas Dewald. Cujo: efficient detection and prevention of drive-by-download attacks. In *Proceedings of the 26th Annual Computer Security Applications Conference*, pages 31–39. ACM, 2010.

[RLZ08] Paruj Ratanaworabhan, Benjamin Livshits, and Benjamin Zorn. Nozzle: A defense against heap-spraying code injection attacks. Technical Report MSR-TR-2008-176, Microsoft Research, 2008.

[Roa07] Joan Robert Roaspana. SHELIA: a client honeypot for client-side attack detection. Master's thesis, Vrije Universiteit Amsterdam, 2007.

[Ros58] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[RPF10] W. Lee R. Perdisci and N. Feamster. Behavioral clustering of HTTP-based malware and signature generation using malicious network traces. In *Proc. of USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 391–404, 2010.

[RSG16] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144. ACM, 2016.

[RSL$^+$10] Konrad Rieck, Guido Schwenk, Tobias Limmer, Thorsten Holz, and Pavel Laskov. Botzilla: Detecting the "phoning home" of malicious software. In *Proc. of 25th ACM Symposium on Applied Computing (SAC)*, pages 1978–1984, March 2010.

[RTWH11] Konrad Rieck, Philipp Trinius, Carsten Willems, and Thorsten Holz. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security*, 19(4):639–668, 2011.

[Sau10] Martin Sauter. *From GSM to LTE: an introduction to mobile networks and mobile broadband.* John Wiley & Sons, 2010.

[SB14] Guido Schwenk and Sebastian Bach. Detecting behavioral and structural anomalies in mediacloud applications. *arXiv preprint arXiv:1409.8035*, 2014.

[SBKR12] Guido Schwenk, Alexander Bikadorov, Tammo Krueger, and Konrad Rieck. Autonomous learning for detection of javascript attacks: vision or reality? In *Proceedings of the 5th ACM workshop on Security and artificial intelligence*, pages 93–104. ACM, 2012.

[Sch15] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[SCW$^+$03] Rong She, Fei Chen, Ke Wang, Martin Ester, Jennifer L Gardy, and Fiona SL Brinkman. Frequent-subsequence-based prediction of outer membrane proteins. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 436–445. ACM, 2003.

[SDHD07] S. Stover, D. Dittrich, J. Hernandez, and S. Dietrich. Analysis of the Storm and Nugache Trojans: P2P is here. *USENIX ;login:*, 32(6):18–27, 2007.

[SGHSV11] Brett Stone-Gross, Thorsten Holz, Gianluca Stringhini, and Giovanni Vigna. The underground economy of spam: A botmaster's perspective of coordinating large-scale spam campaigns. In *Proc. of USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2011.

[SK12] Masashi Sugiyama and Motoaki Kawanabe. *Machine learning in non-stationary environments: Introduction to covariate shift adaptation.* MIT press, 2012.

[SPD$^+$09] Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alex Smola, and S.V.N. Vishwanathan. Hash kernels for structured data. *J. Mach. Learn. Res.*, 10:2615–2637, December 2009.

[SPST$^+$01] B. Schölkopf, J. Platt, J. Shawe-Taylor, A.J. Smola, and R.C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7):1443–1471, 2001.

[SR11] Guido Schwenk and Konrad Rieck. Adaptive detection of covert communication in http requests. In *Computer Network Defense (EC2ND), 2011 Seventh European Conference on*, pages 25–32. IEEE, 2011.

[SRS05a] Sören Sonnenburg, Gunnar Rätsch, and Christin Schäfer. Learning interpretable svms for biological sequence classification. In *Annual International Conference on Research in Computational Molecular Biology*, pages 389–407. Springer, 2005.

[SRS05b] Sören Sonnenburg, Gunnar Rätsch, and Bernhard Schölkopf. Large scale genomic sequence svm classifiers. In *Proceedings of the 22nd international conference on Machine learning*, pages 848–855. ACM, 2005.

[SS02] B. Schölkopf and A.J. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

[SS06] Christian Seifert and Ramon Steenson. Capture – honeypot client (Capture-HPC). Victoria University of Wellington, NZ, `https://projects.honeynet.org/capture-hpc`, 2006.

[SS11] M. Ben Salem and S. J. Stolfo. Decoy document deployment for effective masquerade attack detection. In *Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, pages 35–54, 2011.

[STC04] John Shawe-Taylor and Nello Cristianini. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

[SWS$^+$00] Bernhard Schölkopf, Robert C Williamson, Alex J Smola, John Shawe-Taylor, and John C Platt. Support vector method for novelty detection. In *Advances in neural information processing systems*, pages 582–588, 2000.

[Sym11] Symantec Internet Security Threat Report: Trends for 2010. Vol. 16, Symantec, Inc., 2011.

[TD99] D.M.J. Tax and R.P.W. Duin. Support vector domain description. *Pattern Recognition Letters*, 20(11–13):1191–1199, 1999.

[TD04] David MJ Tax and Robert PW Duin. Support vector data description. *Machine learning*, 54(1):45–66, 2004.

[TK02] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *J. Mach. Learn. Res.*, 2:45–66, March 2002.

[TLLP11] Julie D Thompson, Benjamin Linard, Odile Lecompte, and Olivier Poch. A comprehensive benchmark study of multiple sequence alignment methods: current challenges and future perspectives. *PloS one*, 6(3):e18093, 2011.

[TVLRD17] Niek Tax, Ilya Verenich, Marcello La Rosa, and Marlon Dumas. Predictive business process monitoring with lstm neural networks. In *International Conference on Advanced Information Systems Engineering*, pages 477–492. Springer, 2017.

[ULD16] Merve Unuvar, Geetika T Lakshmanan, and Yurdaer N Doganata. Leveraging path information to generate predictions for parallel business processes. *Knowledge and Information Systems*, 47(2):433–461, 2016.

[VBMKM09] Paul Von Bünau, Frank C Meinecke, Franz C Király, and Klaus-Robert Müller. Finding stationary subspaces in multivariate time series. *Physical review letters*, 103(21):214101, 2009.

[VDAADM$^+$11] Wil Van Der Aalst, Arya Adriansyah, Ana Karla Alves De Medeiros, Franco Arcieri, Thomas Baier, Tobias Blickle, Jagadeesh Chandra Bose, Peter Van Den Brand, Ronald Brandtjen, Joos Buijs, et al. Process mining manifesto. In *International Conference on Business Process Management*, pages 169–194. Springer, 2011.

[VKMG17] Marina M-C Vidovic, Marius Kloft, Klaus-Robert Mueller, and Nico Goernitz. Ml2motif—reliable extraction of discriminative sequence motifs from learning machines. *PloS one*, 12(3):e0174392, 2017.

[WBC10] Sean Whalen, Matt Bishop, and James P Crutchfield. Hidden markov models for automated protocol learning. In *Security and Privacy in Communication Networks*, pages 415–428. Springer, 2010.

[WBH$^+$09] Peter Wurzinger, Leyla Bilge, Thorsten Holz, Jan Goebel, Christopher Kruegel, and Engin Kirda. Automatically generating models for botnet detection. In *Proc. of European Symposium on Research in Computer Security (ESORICS)*, pages 232–249, 2009.

[WBJ$^+$06] Yi-Min Wang, Doug Beck, Xuxian Jiang, Roussi Roussev, Chad Verbowski, Shuo Chen, and Samuel T. King. Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities. In *Proc. of Network and Distributed System Security Symposium (NDSS)*, 2006.

[WCKK08] Gilbert Wondracek, Paolo Milani Comparetti, Christopher Kruegel, and Engin Kirda. Automatic network protocol analysis. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)*, pages 1–18, 2008.

[WDL$^+$09] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 1113–1120, New York, NY, USA, 2009. ACM.

[WM12] Sida Wang and Christopher D Manning. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*, pages 90–94. Association for Computational Linguistics, 2012.

[WPS06] Ke Wang, Janak J Parekh, and Salvatore J Stolfo. Anagram: A content anomaly detector resistant to mimicry attack. In *Recent Advances in Intrusion Detection*, pages 226–248. Springer, 2006.

[WS04] Ke Wang and Salvatore J Stolfo. Anomalous payload-based network intrusion detection. In *Recent Advances in Intrusion Detection*, pages 203–222. Springer, 2004.

[WSAR13] Christian Wressnegger, Guido Schwenk, Daniel Arp, and Konrad Rieck. A close look on n-grams in intrusion detection: anomaly detection vs. classification. In *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*, pages 67–76. ACM, 2013.

[XPK10] Zhengzheng Xing, Jian Pei, and Eamonn Keogh. A brief survey on sequence classification. *ACM Sigkdd Explorations Newsletter*, 12(1):40–48, 2010.

[YHL12] Guo-Xun Yuan, Chia-Hua Ho, and Chih-Jen Lin. Recent advances of large-scale linear classification. *Proceedings of the IEEE*, 100(9):2584–2603, 2012.

[YWGR13] Fabian Yamaguchi, Christian Wressnegger, Hugo Gascon, and Konrad Rieck. Chucky: Exposing missing checks in source code for vulnerability discovery. In *Proceedings of the 20th ACM Conference on Computer and Communications Security*, pages 499–510. ACM, 2013.

[ZP00] Y. Zhang and V. Paxson. Detecting backdoors. In *Proc. of USENIX Security Symposium*, pages 157–170, 2000.

[ZSSL11] Junjie Zhang, Christian Seifert, Jack W. Stokes, and Wenke Lee. ARROW: GenerAting SignatuRes to Detect DRive-By DOWnloads. In *Proc. of the International World Wide Web Conference (WWW)*, 2011.