9 Machine learning for intrusion and anomaly detection

Chapter overview

Chapter summary

An overview of anomaly and intrusion detection (prevention) as well as machine learning is presented along with a discussion of open research problems. Machine learning provides a scalable and decentralized framework for detection, analysis, and decision making in network security. As an example of a distributed machine-learning (ML) scheme, a distributed binary SVM classification algorithm is analyzed. It is then applied to the problem of malware detection in mobile devices using behavioral personalized signatures.

9.1 Intrusion and anomaly detection

Networked systems are difficult to **observe and control** even by their owners due to their **complexity**. The complexity is a result of three factors: powerful and ubiquitous computing hardware, complex software running on the hardware, and interconnectivity between separate computing systems. The same factors also make it difficult to observe the multitude of processes on modern networked systems. As a simple example, it is practically impossible for users to continuously be conscious of which software and communication processes are running on their mobile phone and to control their functionality.

An intuitive result, supported also by the classical control theory, states that it is very difficult to control any system that is not observable. Therefore, **observation** capabilities should be seen as the first step towards controlling networked systems by their owners. Increasing observability of systems (by their owners) enables building better defenses against unwanted behavior and malicious security compromises. Nevertheless, observation capabilities are often not a built-in feature of networked systems. Furthermore, collecting, interpreting, and storing such data brings additional overhead and requires a nontrivial investment.

It is useful to briefly review the history of network security in order to better understand the current problems. When networked systems started playing an increasingly important role in almost all aspects of modern life, security threats against them both increased and were taken more seriously by the owners of the systems. Ever since, the potential and actual costs of security compromises have been naturally motivating defensive measures. The first defenses were mostly static such as firewalls establishing a perimeter and protecting it or reactive such as the signature-based antivirus software. Next, dynamic intrusion detection systems emerged, which increased system observation capabilities. More recently, the trend is towards dynamic and preventive measures with the emergence of intrusion prevention systems augmenting detection capabilities. This is partly a result of technological developments making such preventive measures feasible on the network and hosts.

On the other side, the attacks have also become more sophisticated. The security threats have evolved from simple ones driven by scientific curiosity of renegade enthusiasts to complex and distributed attacks for financial gain, sometimes in connection with organized crime. The botnets which launch hybrid worm-Trojan malware from thousands of computers (without the knowledge of their owners) exemplify the level of sophistication of recent attacks.

The **castle analogy** provides a useful and fun model for visualizing the fundamental security concepts discussed above [69]. If the networked system to be secured is compared to a medieval castle, then static security measures such as firewalls, authentication, and access control correspond to the walls and gates of the castle, respectively. Reactive measures such as antivirus software can be similarly compared to antisiege weapons. All these semistatic security measures naturally require maintenance and upgrades in the face of new technologies. On the other hand, no matter how strong the walls are, the castle is not secure without guards defending it. Without dynamic measures the castle, however well protected, would be vulnerable and “blind” to security violations and attacks. The direct counterpart of these guards in the IT context is intrusion detection systems. Similarly, scouts and guards patrolling the vicinity of the castle can be compared to intrusion prevention systems which detect and try to address threats before they arrive at the gates.

The characteristics of attackers and defenders can also be modeled within the same castle analogy. Over time, the defensive guards have evolved from simple people or owners defending the castle to professional soldiers protecting it. The attackers, likewise, turned into well-organized and equipped opponent armies from their humble beginnings of disorganized roving bands of outlaws.

This chapter focuses on computer-assisted security attack detection and analysis using (statistical) ML methods from the perspective of the defense side. As already mentioned, observation capabilities provide the necessary basis for attack prevention and security measures. It is obvious yet worth mentioning that observation and detection in the context of network security cannot be done manually by human beings and require computer assistance at the minimum. Therefore, *state-of-the-art ML techniques are potentially an indispensable component of computer assistance systems for the detection, analysis, and prevention of security threats*.

9.1.1 Intrusion detection and prevention systems

Intrusion detection can be defined as the process of monitoring the events occurring in a computer system or network and analyzing them for signs of intrusions [25, 26]. This classical definition is somewhat limited and focuses mainly on detecting attacks after they occur and reacting to them. Recent developments in the field have resulted in a more extended approach of intrusion prevention where monitoring capabilities are utilized to take preventive actions to defend against malicious attacks before or as they occur.

The evolution of intrusion detection systems to hybrid intrusion detection and prevention systems is not very surprising in light of the close relationship between observation and control as discussed above. While intrusion detection focuses more on detection and reporting aspects of the problem, prevention systems tend to be more action- and response-oriented. Responses such as filtering of malicious packets at the perimeter of the network aim to prevent attacks proactively. This shift can be seen as a result of continuous technological improvements, which enable better and economically feasible defensive solutions. Consequently, once monitoring capabilities are developed and available, it makes sense to deploy them both inside a networked system and at the periphery.

Although IDPSs are constantly evolving, it is useful to study the basic concepts and building blocks, which mostly remain invariant. An IDPS consists of three main components: *information sources*, *analysis*, and *response*.

The **information sources** in an IDPS observe the networked system and collect data that will help to detect and prevent attacks. They can be implemented, for example, as software agents running as virtual sensors or on dedicated hardware for packet inspection. The data collected is often sequential and heterogeneous in nature. Depending on the type and number of sources, the amount of data collected can be quite large, which may require significant storage and processing capabilities.

A set of (virtual) sensors network can be deployed as part of an IDPS in order to collect information and detect malicious attacks on the network (see Figure 9.1). A virtual sensor network is defined as a collection of autonomous software agents that monitors the system and collects data for detection purposes. The sensors report possible intrusions or anomalies occurring in a subsystem using common pattern recognition and statistical analysis techniques. Some of the desired properties of sensors can be summarized [191] as

- **completeness:** a sensor should cover the assigned subsystem in the sense that it collects all the information necessary for detection. Structural differences between the information sources such as log files, packet headers, etc. and resource constraints increase the difficulty of this task;
- **correctness:** the integrity of the collected information by the sensor should be ascertained against any tampering by the attacker. Misinformation is much more harmful to the IDS than partial information;
- **resource usage:** especially host-based sensors use a portion of the system resources, which include memory, storage space, processing power, and bandwidth. Hence, they are naturally bounded by them;
- **reconfigurability:** sensors should be reconfigurable both in terms of operating parameters and deployment.

It is informative to compare and contrast the virtual sensors, including those in dedicated appliance devices, with physical hardware-based sensors (motes). Motion, temperature, and biometric sensor motes are, for example, utilized for the physical security of buildings and rooms. While both classes of sensor are functionally similar with the common goal of collecting information on a system, there are some fundamental differences in terms of resource constraints. For example, there is no limit to the number of virtual sensors to be deployed in a system except from the communication and computation overhead. Clearly, this is not the case with physical sensors. Moreover, virtual sensors do not have power constraints. On the other hand, communication overhead is a problem for both networks. The unique characteristics of the virtual sensor network have to be taken into account in the IDS deployment process as well as in addressing the related resource allocation problems.

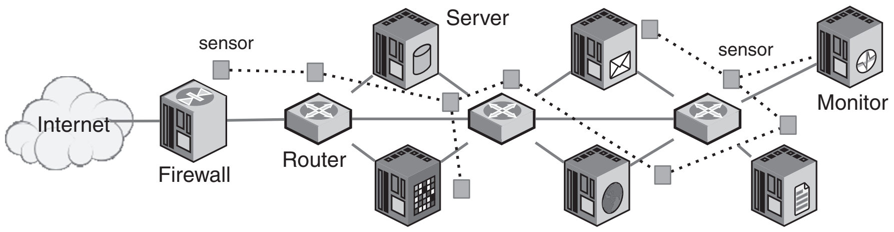


Figure 9.1 Graphical representation of a virtual sensor network for intrusion detection.

According to their deployment and source of information, IDPSs are categorized as host-based or network-based. **Network-based** IDPSs detect attacks by capturing and analyzing individual packets on the network. They can be deployed at the periphery or gateway, monitoring the entire traffic or at individual switches. They can provide blanket protection to the entire network and potentially stop attacks before they reach the protected system. On the other hand, large data volumes or encrypted traffic pose significant challenges for such systems.

A concept related to information collection is a **honeypot**. A honeypot is an isolated, unprotected, and monitored component of the protected system, e.g. a computer. Its sole purpose is to collect attack data but it has no other production value. The information collected by honeypots is valuable as a surveillance and early warning tool. Honeypots can be deployed in production environments to assess the actual threats faced as well as for research purposes.

Host-based IDPSs are deployed in individual computer systems and utilize, for example, operating system calls and system logs to detect attacks. They can also check file system integrity. Although they are immune to the issues faced by network-based variants, host-based IDPSs have their own problems. They are often difficult to manage, have nontrivial computational overhead, and can themselves be targeted by attacks.

Once the observational data is collected, it needs to be analyzed for signs of security threats. There are two main approaches for the **analysis** of the collected data: misuse and anomaly detection. **Misuse detectors** analyze the data, looking for events that match a predefined pattern of a known attack. Since the patterns corresponding to known attacks are often called (attack) signatures, the terms "*signature-based detection*" and "misuse detection" are used interchangeably. The most well-known and widely used example of signature-based detection is that used in antivirus software. Another example is the rules in the Snort IDPS software.[1] The signatures have to be written manually by experts one by one for each threat. The updated signatures are then sent to individual IDPS software periodically. As a countermeasure the attackers create multiple variants of the same malware as well as malware that self-mutates. Zero-day attacks for which no signature is generated yet and the huge number of threat types make signature-based detection a limited solution.

The second analysis approach is **anomaly detection** where "abnormal" or unusual behavior (anomalies) on the networked system are identified using various pattern recognition and (statistical) machine-learning techniques. The underlying assumption here is that attacks will be observable in the data collected by (legitimate) activity differing from "normal." This approach has the potentially desirable properties of scalability and detection of previously unknown attacks for which there is no manually generated signature. In practice, however, anomaly detectors have often very high false positive rates, i.e. mistake legitimate events for anomalies. This is maybe not surprising considering that detecting an anomaly in a large dataset is roughly analogous to

[1] http://www.snort.org/

Table 9.1 Comparison of signature-based and anomaly detection

Signature-based	Anomaly
Effective in detecting known attacks	Detects previously unknown attacks
Quick and reliable	Many false-alarms
Manual signature generation	Often requires training sets
Not scalable with threat types	Potentially scalable

finding a needle in a haystack. As a further complication, most ML techniques require a “clean” training set to learn normal behavior, which may not exist at all. A summary of signature-based and anomaly detection techniques’ properties is shown in Table 9.1.

Once the data is collected and analyzed, the **response** to detected incidents can be active or passive. **Passive response** mainly involves alerting the system administrators and logging the suspected events for later analysis or network forensics. The problems of prioritizing alerts and how to present them to the administrators can be addressed within an optimization framework as discussed in Section 8.1.2. Given the limited time and attention of security officers, these tasks have to take usability into account as an important factor.

Active response schemes are one of the main distinguishing characteristics of intrusion prevention systems. One possible active response approach is to drop suspected packets and flows to prevent attacks before they reach the protected system. If not carefully configured, however, such measures can have significant detrimental side effects. A less drastic approach is to initiate additional investigation on the incident, for example, by collecting more data than usual via additional sensors. Another alternative is to reconfigure the environment, e.g. by increasing access restrictions and isolating parts of the protected system. Active response is crucial especially in cases where *response time* is an important factor such as fast-spreading worms or viruses which can paralyze the whole networked system within minutes.

9.1.2 Open problems and challenges

Although IDPSs are important components of network security, they are far from perfect and face open problems. One problem is the attacks targeting IDPSs and their capabilities directly. A second one is usability, deployment, and configuration of IDPSs. A third problem is the inherent difficulty of the detection task itself, which, using the earlier analogy, is similar to “searching for a needle in a haystack.”

Just like the system it protects, the **IDPS can itself be targeted** by attackers. Carefully constructed packets or behavior patterns may function as a DoS attack against the IDPS and render it ineffective. Alternatively, the attackers can use clever stealth techniques to avoid detection by IDPS. One example is polymorphic worms that change their representation as they spread throughout networks and can evade especially signature-based worm detection systems which rely on fixed descriptions. Another example is encrypted network traffic that is becoming more and more common. If an attacker encrypts the command and control mechanisms of a botnet, the challenge for an IDPS to detect and disrupt it becomes an order of magnitude more difficult. Another class of attacks targets the learning phase of anomaly detection schemes. Here, a malicious adversary constructs labeled samples artificially on purpose and lets them be observed by the IDPS. If these samples are used by the detection algorithm as data points during training, the accuracy of the classifier is compromised for the benefit of the attacker.

Usability of IDPSs is an important problem that is usually overlooked by security researchers, partly owing to its nontechnical nature. Since each networked system is different, the IDPS has to be configured according to the deployment environment. Consequently, the initial deployment and configuration of an IDPS in a medium- to large-scale networked system can be so complex that it becomes a barrier itself. Recently, a variety of security appliances have been developed as a solution to address the usability issue. They are, however, usually only network-based and focus mainly on packet filtering.

A third problem, which may well be the most significant of all, is the **base-rate fallacy** and its implications for intrusion detection [24]. It is a direct consequence of the basic Bayesian theorem and is best explained by means of a numerical example. If an IDPS generates one million records a day and only 20 of them indicate real intrusions, then to reach a true positive (attack detection) rate of 0.7 and a Bayesian detection rate (the rate an alarm really indicates an intrusion) of 0.58, the false-alarm probability has to be as low as 0.00001.

This rather unexpected result puts significant constraints on the desired effectiveness of an IDPS. One of the underlying issues here is the rarity of intrusion-related events within a significant amount of data. Another is the adverse effects of false-alarms, which make the job of security officers more difficult and potentially waste resources. In other words, the situation is worse than looking for a "needle in a haystack" since one has not only to go through a big haystack to find the needle but also have a very sharp eye distinguishing each straw from the needle very accurately.

9.2 Machine learning for security: an overview

Complex networked systems cannot be continuously observed, controlled, and defended against security compromises *manually* by their users or administrators. **Computer assistance** is absolutely necessary to facilitate all these processes. This requires certain decisions to be taken automatically without human intervention. Machine (statistical) learning studies such decision-making problems in a variety of fields ranging from computer vision to speech recognition. Hence, machine learning has significant potential and has been increasingly used in the security domain.

The application of ML to network security has interesting implications for both fields. While ML creates a foundation for computer-assisted decision making in network security, it in turn benefits from unique aspects of this relatively novel application domain. This opens up interesting research directions such as data mining for security, distributed ML, and adversarial ML. Among these, distributed machine learning in networked environments will be discussed in Section 9.3 in more detail.

As we have already seen, intrusion and anomaly detection involves collecting a significant amount of observation data and its analysis. Therefore, it is natural to adopt existing statistical and pattern recognition schemes in the analysis of the collected data to classify anomalous and potentially malicious behavior. One way to approach this problem is to characterize the data within a parametric statistical model and estimate its parameters from the dataset. The anomalies potentially indicating malicious behavior can then be distinguished using hypothesis testing. There exists a rich literature on detection and estimation theory which approaches this problem from a signal processing and information theoretic perspective. This point of view will be presented and discussed in Chapter 10.

On the other hand, it is often difficult to characterize the IDPS-collected data by a probability distribution or an independent identically distributed random process. In such cases nonparametric ML and data-mining algorithms such as clustering, support vector machines (SVMs), or kernel-based methods become more relevant. It is worth noting that the distinction between parametric (statistical) and nonparametric (ML) methods is not as crisp as it may first seem. Both approaches aim to solve similar problems, share fundamental properties, and usually borrow many techniques from each other.

Although it is not feasible to provide here a comprehensive treatment of ML methods, a brief overview is beneficial to better understand their advantages and limitations within network security context. The basic properties of various ML algorithms [40] and their relevancy to decision making in the security domain are discussed next.

9.2.1 Overview of machine-learning methods

We provide in this section an overview of preprocessing, unsupervised and supervised learning, and reinforcement learning, which are the salient ML methods.

Preprocessing

The first step of data analysis is *preprocessing* where original input variables are transformed into a new variable space and – if possible – undergo a dimension reduction to facilitate subsequent computations. This may involve *feature extraction* where a subset of features that preserves useful discriminatory information are selected. The data transformation and feature selection may be done in such a way that variance of the projected feature set is maximized as in the well-known principle component analysis (PCA). Preprocessing is one of the most important yet sometimes overlooked aspects of ML and plays a crucial role in the success of any subsequent data analysis method.

In the case of network security problems, the amount of data collected by IDPS is often large. Furthermore, the data is usually noisy and redundant due to its observational nature. Proper application of preprocessing techniques in combination with problem-dependent domain knowledge helps to reduce the problem dimension and decrease the computational costs in subsequent steps significantly. However, proper care must be shown in preprocessing in order *not* to discard potentially useful information from original input data.

Unsupervised learning

One of the two main approaches in the analysis of IDPS observations is **unsupervised learning**. Clustering, density estimation, and visualization are among the well-known unsupervised learning techniques. *Clustering* discovers patterns in the dataset by grouping data points with similar features, *density estimation* studies the distribution of the data within the input space, and *visualization* projects the data into two or three dimensions to facilitate its human readability.

Security visualization[2] aims to process and transform complex security data collected into human-readable visual representations. Hence, the actual tasks of intrusion and anomaly detection are left to system administrators to be handled manually. The main idea is to utilize the superb pattern recognition capabilities of the human brain instead of relying on various ML algorithms. Thus, security visualization can be seen both as a computer-assistance scheme for security monitoring and as an alternative approach that addresses the disadvantages of fully automated anomaly detection methods. On the other hand, this approach has its own potential disadvantages such as imperfect data aggregation and scalability issues owing to its manual nature.

Density estimation assumes a probabilistic model of the input data and estimates the parameters of the model, e.g. probability distribution, from the given dataset which may be incomplete. Therefore, it can be categorized as a parametric or a statistical technique. As an example, the expectation–maximization algorithm is a well-known formal statistical technique for finding the maximum likelihood estimates of the model parameters. It can be used in the security context, for example, to characterize network traffic, and hence detect flow anomalies which may indicate DoS attacks.

Clustering is probably the most well-known and widely used unsupervised learning method in ML and data mining. Clustering techniques categorize data points in the input set to subsets or categories (clusters) based on a similarity measure defined over their feature space. A well-known variant called *K-means clustering* partitions n given points into a predefined k cluster in which each point is assigned to the cluster with the nearest mean value. Expectation–maximization is another widely used clustering method. While neither algorithm guarantees a globally optimal solution, they are both fast and easy to implement.

Clustering techniques have been utilized in network security especially for anomaly detection. The common approach is as follows: the data collected by IDPS is clustered based on a chosen feature vector and distance metric on the feature space. The computed clusters are then used to characterize the data. For example, the "neurons" of a SOM are a discretized representation of the input data. After this analysis step, subsequent

[2] http://www.secviz.org/

IDPS observations can be compared with the computed representation using a distance metric. If it can be ensured that the initial dataset represents "normal" system behavior, then any future data that significantly deviates from the computed representation can be interpreted as an "anomaly."

Supervised learning

ML methods that aim to "learn" a function from a given *training data* set constitute the second main approach, called **supervised learning**, for the analysis of IDPS observations. The objective of *classification* problems is to learn a function that assigns a label from a discrete set such as $\{clear, malicious\}$ to each data point. If the label set has only two elements, then the problem is called binary classification. On the other hand, if the function to be learned is real-valued, then the problem is called *regression*. For example, each packet analyzed by the IDPS can be assigned a real-valued suspicion level between zero and one.

Two main criteria for the success of supervised learning algorithms are *prediction* and *generalization*. The learned function should be able to predict successfully the output for data other than in the training set used. At the same time, it should act as a model that captures the underlying characteristics of the training data and generalize to new data points. The opposite behavior is called *overfitting* where the function describes the training data perfectly but has poor predictive power. The performance of a supervised learning algorithm is quantitatively assessed using a *test data* set and *cross-validation* techniques.

Supervised ML techniques offer an invaluable set of formalized computing methods to develop computer-assisted detection, analysis, and decision systems for network security. These methods have been successfully applied to a wide variety of fields ranging from image recognition, speech processing, and data mining. In the network security domain, whether the objective is to distinguish whether a packet contains malicious payload or to assess how suspicious is certain behavior (e.g. flow pattern), the problem can be naturally formulated within the ML framework. IDPSs benefit from the scalability and computational power of ML algorithms as opposed to manual processing by system administrators. Furthermore, an IDPS usually generates significant amounts of data which improves the performance of many ML algorithms.

9.2.2 Open problems and challenges

In addition to its potential, the application of machine learning to network security gives rise to multiple problems and research directions. The first issue is the lack of good quality datasets to assess the performance of ML algorithms in the security domain. This lack of data hinders research efforts and is in stark contrast to other application fields of ML such as computer vision. The second problem is the fact that attackers can (secretly) influence the learning and decision processes in ML, especially in the training phase. This opens up an interesting new research direction for ML called *adversarial machine learning*. The adversarial ML algorithms, which adopt the role of the defender, try to accomplish their objectives while facing a malicious attacker who aims to undermine their efforts. The third problem is *distributed machine learning*, which extends ML to networked environments where the training set itself may be distributed and the communication between distributed algorithm subroutines may be imperfect.

The current **lack of good quality datasets** in network security is an important yet hopefully temporary problem. The problem was acknowledged a decade ago and the famous IDS evaluation dataset was published by the Defense Advanced Research Projects Agency (DARPA)[3] in 1998, which was then used in 1999 at the knowledge discovery and data mining (KDD) cup. Since then, the KDD dataset has been used by numerous papers, partly owing to lack of alternatives and partly for convenience. Recently, various flaws with this dataset have been discovered. Its usage as an evaluation tool is discouraged by many experts. Disappointingly, there is almost no other dataset proposed as a replacement until now.

Some security researchers claim that the only way of checking a security mechanism is to test it experimentally in a realistic setting. Although it has its obvious merits, this approach can also be potentially harmful to research efforts in the field. Building an experimental IDPS environment is a costly endeavor in terms of both manpower and monetary expenses. Therefore, such an approach creates a significant barrier of entry for cross-disciplinary efforts and prevents knowledge transfer from the ML community to security researchers and vice versa. It is unrealistic to expect every ML researcher, who is interested in security as an application field, to have a full-fledged and expensive security laboratory. Considering that quantitative evaluation using training and testing datasets is an indispensable aspect of ML algorithms, lack of high-quality datasets shared by the whole community is potentially harmful to the healthy development of ML in the security domain.

Adversarial machine learning constitutes a second open problem and potentially promising research direction. In classical ML, the training data can be noisy; yet there is no conscious attacker trying to manipulate it. The presence of an attacker, who either tries to hide attack symptoms or poison the training data to mislead the algorithm, brings a novel and interesting dimension to existing ML approaches discussed above. Given the widespread use of optimization formulations in classical ML, *game theory*, which can be seen as the multi-person decision-making counterpart of optimization, can provide the needed framework to investigate adversarial ML problems. Security game formulations discussed in Chapter 3 may be especially relevant here.

Recent advances in cloud computing, multiprocessor systems, and multicore processors make **distributed machine learning** an important research field relevant not only to the security domain but also to many others. The widespread and cheap availability of powerful parallel and distributed computing resources already provides a motivation for distributed and parallel ML schemes. They find a natural application domain in network security due to the inherently distributed nature of networked systems. The data collection, analysis, and storage processes of a networked IDPS are often decentralized. Distributed ML can easily be deployed within such an IDPS and remove the need

[3] DARPA is an agency of the United States Department of Defense.

for forced centralization at a security center within the network. A distributed binary classification algorithm will be discussed in the next section as an example.

9.3 Distributed machine learning

Powerful parallel and distributed computing systems have recently become widely available in the form of multicore processors and cloud computing platforms. Motivated by these developments, **distributed machine learning** algorithms can be naturally applied to address various decision-making problems in network security, where data collection and storage processes are inherently distributed. Both the efficiency and robustness of IDPSs are improved by taking advantage of computing capabilities distributed over the whole networked system. Furthermore, overall communication overhead is decreased by not sending all of the observation data to a single information fusion and decision-making center. A similar approach is also applicable in multicore processors or GPU programming, albeit in a smaller scale.

This section discusses a **distributed SVM-based binary classification framework** as an exemplary distributed ML scheme. First, the quadratic SVM binary classification problem is divided into multiple separate subproblems by relaxing it using a penalty function. Then, distributed continuous- and discrete-time gradient algorithms are analyzed that solve the relaxed problem iteratively and in parallel. A sufficient condition is derived under which the synchronous parallel update converges to the approximate solution geometrically. Next, an asynchronous algorithm is investigated where only a random subset of processing units update at a given time instance and show its convergence under the same condition. Subsequently, stochastic update algorithms are studied which may arise due to imperfect information flow between units or distortions in parameters. Sufficient conditions are derived under which a broad class of stochastic algorithms converge to the solution.

Unlike sequential or centralized approaches in the classical ML literature, the focus here is exclusively on **parallel update schemes** to address classification problems such as detecting malware and suspicious activity given data obtained by an IDPS. The approach introduced allows individual processing units to do simultaneous computations. This is in contrast to training SVMs either sequentially or in parallel first and then fusing them into a centralized classifier for intrusion detection. The distributed SVM classifier is especially useful when each unit has access to a different subset of the overall dataset, which may be time-varying, and has its own computing resources.

In practice, the communication overhead and the number of support vectors resulting from the relaxation of the original binary SVM classification problem can be unacceptably high, regardless of the update algorithm used. To address this issue, **active set methods** are utilized, which have been widely used in solving general quadratic problems as well as in centralized SVM formulations. The resulting algorithm is greedy in nature and has been observed to converge to a solution in a low number of rounds. In addition, the framework is suitable for online learning and gives a choice on the upper-bound of the resulting support vectors.

9.3.1 SVM classification and decomposition

First, an overview of the classical SVM-based binary classification problem is provided. Subsequently, a relaxation of the centralized formulation and its decomposition into parallel subproblems are discussed.

Centralized classification problem

Assume that a set of labeled data,

$$S := \{(x_1, y_1), \ldots, (x_N, y_N)\},$$

is given where $x_d \in \mathbb{R}^L$ and $y_d \in \{\pm 1\}$, $d = 1, \ldots, N$. A *classification problem* is considered with the objective of deriving a generalized rule from this *training data* that associates an input x with a label y as accurately as possible. It is important to note that no assumption is made on the nature of the training or test data. A well-known method for addressing this binary classification problem involves the representation of input vectors in a high(er)-dimensional feature space through a (nonlinear) transformation. Define the dot product of two feature vectors, say x_d and x_e, as a *kernel*, $k(x_d, x_e)$. In many cases this dot product can be computed without actually calculating the individual transformations, which is known as the *kernel trick*.

The optimal margin nonlinear binary **SVM classification** problem [162] described above is formalized using the quadratic programming problem

$$\max_{\alpha_d} \sum_{d=1}^{N} \alpha_d - \frac{1}{2} \sum_{d=1}^{N} \sum_{e=1}^{N} \alpha_d \alpha_e q_{d,e}$$

$$\text{such that } \alpha_d \geq 0, \ d = 1, \ldots, N \tag{9.1}$$

$$\text{and } \sum_{d=1}^{N} \alpha_d y_d = 0, \tag{9.2}$$

where the α_d are the Lagrange multipliers of the corresponding *support vectors* (SVs) and $q_{d,e}$ are the entries of the positive definite matrix

$$Q := \left[(y_d y_e k(x_d, x_e))_{d,e} \right]_{N \times N}. \tag{9.3}$$

The positive definiteness of Q simply follows from the assumption that the kernel matrix $K_{d,e} := [k(x_d, x_e)]_{N \times N}$ is positive definite [162]. The decision function classifying an input x is then

$$f(x) = \operatorname{sgn}\left(\sum_{l=1}^{N_D} \alpha_l y_l k(x, x_l) + b \right),$$

where b is the bias term and $N_D < N$ is the number of support vectors. Here, the well-known *representer theorem* [162] enables a finite solution to the infinite-dimensional optimization theorem in the span of N particular kernels, $k(x, x_d)$, centered on the training points x_d, $\forall d$.

Decomposition into subproblems

In order to decompose the centralized classification problem into **subproblems**, define a set $\mathcal{M} := \{1, 2, \ldots, M\}$ of separate *processing units* with access to different (possibly overlapping) subsets, $\mathcal{S}_i$, $i \in \mathcal{M}$, of the labeled training data such that $\mathcal{S} = \bigcup_{i=1}^{M} \mathcal{S}_i$. The initial partition of the data $\mathcal{S}$ can be due to the nature of the specific problem at hand or as a result of a partitioning scheme at the preprocessing stage. Given the partition, define the vectors $\{\alpha^{(1)}, \alpha^{(2)}, \ldots, \alpha^{(M)}\}$ where the i-th one is $\alpha^{(i)} := \left[\alpha_1^{(i)}, \ldots, \alpha_{N_i}^{(i)}\right]$. In order to devise a distributed algorithm that solves the optimization problem (9.1), it is relaxed by substituting the constraint $\sum_{d=1}^{N} \alpha_d y_d = 0$ by a quadratic penalty function, $0.5M\beta \left(\sum_{d=1}^{N} \alpha_d y_d\right)^2$, where $\beta > 0$. Next, an upper-bound α_{max} on α_d is imposed such that $\alpha_d \le \alpha_{max}\ \forall d$. This upper-bound can be chosen to derive a soft margin hyperplane (i.e. maximizing the margin). Alternatively, it can be chosen as very large to minimize the training error ignoring the margin. Thus, the following constrained optimization problem

$$\max_{\alpha_d \in [0, \alpha_{max}]} F(\alpha) = \sum_{d=1}^{N} \alpha_d - \frac{1}{2} \sum_{d=1}^{N} \sum_{e=1}^{N} \alpha_d \alpha_e q_{d,e} - \frac{M\beta}{2} \left(\sum_{l=1}^{N} \alpha_l y_l \right)^2 \tag{9.4}$$

approximates (9.1). Note that the objective function $F(\alpha)$ is strictly concave in all its arguments due to Q in (9.3) being positive definite, and the constraint set $X := [0, \alpha_{max}]^N$ is convex, compact, and nonempty.

The convex optimization problem (9.4) is next **partitioned** into M subproblems. Hence, the i-th unit's optimization problem is

$$\underset{\alpha_d \in [0, \alpha_{max}],\, d \in \mathcal{S}_i}{\text{maximize}} F_i(\alpha) = \sum_{d \in \mathcal{S}_i} \alpha_d - \frac{1}{2} \sum_{d \in \mathcal{S}_i} \sum_{e=1}^{N} \alpha_d \alpha_e q_{d,e} - \frac{\beta}{2} \left(\sum_{l=1}^{N} \alpha_l y_l \right)^2 . \tag{9.5}$$

Again, the individual objective functions $F_i(\alpha)$ are strictly concave in all $\alpha_d \in \mathcal{S}_i$, and the respective constraint sets are convex, compact, and nonempty for all i.

Remark 9.1 *The individual optimization problems of the units are interdependent. Hence they cannot be solved individually without deployment of an information exchange scheme between the processing units.*

9.3.2 Parallel update algorithms

Continuous-time gradient algorithm

A distributed continuous-time algorithm, similar to that in reference [15], solves the problem (9.4). Clearly, solving all unit problems at the same time is equivalent to finding the solution of the relaxed problem (9.4). One possible way of achieving this objective is to utilize a gradient algorithm that converges to the unique maximum, α^*, of (9.4) which closely approximates that of the original optimization problem. Then, every unit implements the following **gradient algorithm** for each of its training samples

$$\frac{d\alpha_d}{dt} = \kappa_d \left[1 - \frac{\alpha_d q_{dd} - \sum_e \alpha_e q_{d,e}}{2} - \beta y_d \sum_{l=1}^{N} \alpha_l y_l \right]^P \tag{9.6}$$

for all $d \in S_i$ and all i where $\kappa_d > 0$ is a unit-specific step-size constant. Here, the shorthand $[\cdot]^P$ refers to the following projection of the $\dot{a}_d := d\alpha_d/dt$:

$$\dot{\alpha}_d = \begin{cases} \dot{\alpha}_d & \text{if } 0 < \alpha_d < \alpha_{max} \\ \dot{\alpha}_d & \text{if } \alpha_d = 0,\, \dot{\alpha}_d > 0 \text{ or } \alpha_d = \alpha_{max},\, \dot{\alpha}_d < 0 \\ 0 & \text{otherwise} \end{cases}$$

Each unit i has access to only its own (possibly overlapping) **training data** S_i whereas the algorithm (9.6) requires more information to be shared between units. One possible solution to this communication problem is to define a *system node* facilitating the information flow between individual units by collecting and sending back all the necessary information from and to each unit, respectively. Every unit sends to the central node its own set of SVs $D^{(i)} = \left\{ (x_d, y_d, \alpha_d) | x_d, y_d \in S_i, \forall d \text{ s.t. } \alpha_d^{(i)} > 0 \right\}$. The system node aggregates this information $D := \bigcup_{i=1}^{M} D^{(i)}$ to obtain

$$\begin{aligned} x_D &= \{x_d \in S | \alpha_d > 0\} \\ y_D &= \{y_d \in S | \alpha_d > 0\} \\ \alpha_D &= \{\alpha_d > 0\}. \end{aligned}$$

Subsequently, the system node broadcasts the triple (x_D, y_D, α_D) back to all units $\{1, \ldots, M\}$ after which each unit i locally computes the terms

$$\begin{aligned} t^{(i)} &= Q^{(i)}_{N_i \times N_D} \alpha_D \\ u &= \alpha_D^T y_D, \end{aligned} \tag{9.7}$$

where $[\cdot]^T$ denotes transpose operation, N_D is the number of elements of x_D, and $Q^{(i)}_{N_i \times N}$ is the portion of the matrix Q relevant to the unit. Thus, the **unit update algorithm** (9.6) is redefined as

$$\dot{\alpha}_d = \kappa \left[1 - \frac{1}{2}\alpha_d q_{dd} - \frac{1}{2} t_d^{(i)} - \beta y_d u \right]^P, \forall d \in S_i,\ \forall i. \tag{9.8}$$

Theorem 9.2 *The distributed scheme (9.6) globally asymptotically converges to the unique maximum of the centralized problem (9.4).*

Proof In order to investigate the convergence properties of the unit algorithms (9.6), first define a Lyapunov function V on the compact and convex constraint set $X = [0, \alpha_{max}]^N$:

$$V(\alpha) = \sum_{d=1}^{N} \alpha_d - \frac{1}{2} \sum_{d=1}^{N} \sum_{e=1}^{N} \alpha_d \alpha_e q_{d,e} - \frac{\beta}{2} \left(\sum_{l=1}^{N} \alpha_l y_l \right)^2. \tag{9.9}$$

This function is strictly concave and admits its global maximum at the solution of (9.4). Furthermore, its unique maximum, $\alpha^* \in X$, coincides with that of the centralized problem (9.4).

Taking the derivative of the **Lyapunov function** with respect to α_d and time t, respectively, yields

$$\frac{\partial V(\alpha)}{\partial \alpha_d} = 1 - \alpha_d q_{dd} - \frac{1}{2}\sum_{e \neq d} \alpha_e q_{d,e} - \beta y_d \sum_{l=1}^{N} \alpha_l y_l$$

and

$$\frac{dV(\alpha)}{dt} = \sum_{d=1}^{N} \frac{\partial V(\alpha)}{\partial \alpha_d} (\dot{\alpha}_d).$$

If $\alpha_{max} > \alpha_d > 0$ for some d, then

$$\frac{\partial V(\alpha)}{\partial \alpha_d} \dot{\alpha}_d = \kappa_d \left[1 - \alpha_d q_{dd} - \frac{1}{2}\sum_{e \neq d} \alpha_e q_{d,e} - \beta y_d \sum_{l=1}^{N} \alpha_l y_l \right]^2 > 0.$$

In the case of $\alpha_d = 0$ for a data point d, either $\dot{\alpha}_d > 0$ and $(\partial V/\partial \alpha_d)\dot{\alpha}_d > 0$ as above or $\dot{\alpha}_d = 0$ leading to $(\partial V/\partial \alpha_d)\dot{\alpha}_d = 0$. The cases where $\alpha_d = \alpha_{max}$ (i.e. upper-bound) are handled similarly.

If the **trajectory** hits the boundary (i.e. $a_d = 0$ or $\alpha_d = \alpha_{max}$ for some d) at a point other than the unique maximum $\alpha^* \in X$, then $dV(\alpha)/dt > 0$ until the trajectory reaches α^*, where $[dV(\alpha)/dt]^P = 0$. Assume otherwise, i.e. that there exists a point $\tilde{\alpha}$ on the boundary of the set X and $\tilde{\alpha} \neq \alpha^*$. Then, the projection of the gradient $[dV/dt]^P$ is zero at the point $\tilde{\alpha} \in X$. However, due to the strict concavity of V, there exists at least one term d such that $(\partial V/\partial \tilde{\alpha}_d)\dot{\tilde{\alpha}}_d > 0$ resulting in $[dV/dt]^P > 0$ which leads to a contradiction. Thus, the system convergences globally asymptotically to the unique maximum α^*. □

Discrete-time gradient projection algorithm

We discuss a discrete-time counterpart of the parallel update scheme (9.6). In this case, every processing unit implements the following **discrete-time gradient projection** algorithm for each of its training samples

$$\alpha_d(n+1) = [\alpha_d(n) + \kappa_d G_d(\alpha)]^+ \ \forall d, \tag{9.10}$$

where

$$G_d(\alpha(n)) := 1 - \frac{1}{2}\left(\alpha_d(n) q_{dd} - \sum_{e=1}^{N} \alpha_e(n) q_{d,e} \right) - \beta y_d \sum_{l=1}^{N} \alpha_l(n) y_l \,.$$

Here, n denotes the update instances and the notation $[\cdot]^+$ represents the orthogonal projection of a vector onto the convex set X defined by $[\alpha]^+ := \arg\min_{z \in X} \|z - \alpha\|_2$, where $\|\cdot\|_2$ is the Euclidean norm. In this special case, the projection of α_d onto X can be computed by mapping α_d onto $[0, \alpha_{max}]$ for each d. This facilitates an easy implementation of the parallel algorithm.

The function $F(\alpha)$ in (9.4) is a polynomial and hence continuously differentiable in its arguments. Furthermore, there exists a scalar constant C such that

$$\|\nabla F(\gamma) - \nabla F(\delta)\|_2 \leq C\,\|\gamma - \delta\|_2\,,\ \forall \gamma, \delta \in X,$$

where ∇F is the gradient operator. Define $z := \gamma - \delta$. Then,

$$\|\nabla F(\gamma) - \nabla F(\delta)\|_2^2 = z^T A^T A\, z,$$

where

$$A := \frac{1}{2}\mathrm{diag}(Q) + \frac{1}{2}Q + \beta y y^T. \tag{9.11}$$

The matrix $\mathrm{diag}(Q)$ contains the diagonal elements of Q with all its off-diagonal elements set to zero. The scalar constant C is then given by

$$C = \sqrt{\max \lambda(A^T A)},$$

where $\max \lambda(\cdot)$ is the maximum eigenvalue. Therefore, the gradient of the objective function $F(\alpha)$, ∇F, is Lipschitz continuous. Moreover, it is bounded from above on X.

Theorem 9.3 *The gradient projection algorithm (9.10) converges to the unique maximum, α^*, of the objective function F in (9.4), if the step-size constant κ_d satisfies*

$$0 < \kappa_d < \frac{2}{\sqrt{\max \lambda(A^T A)}},\ \forall d.$$

Proof The proof follows directly from the upper-boundedness and strict concavity of the function F and Lipschitz continuity of its gradient ∇F. Propositions 3.3 and 3.4 in reference [38, p. 213] contain the details. □

Theorem 9.4 *Assume that the conditions in Theorem 9.3 hold. Then, the gradient projection algorithm (9.10) converges to the unique maximum α^* of (9.4) geometrically.*

Proof This result follows directly from Proposition 3.5 of [38, p. 215], if there exists a constant $c > 0$ such that

$$(\nabla F(\gamma) - \nabla F(\delta))^T (\gamma - \delta) \geq c\,\|\gamma - \delta\|_2^2,\ \forall \gamma, \delta \in X.$$

Let us again define $z := \gamma - \delta$. Then,

$$\begin{aligned}(\nabla F(\gamma) - \nabla F(\delta))^T (\gamma - \delta) &= z^T A^T z\\ &\geq z^T \max \lambda(A^T) z,\end{aligned}$$

$\forall \gamma, \delta \in X$, where the matrix A is defined in (9.11). We note that the matrix A is the sum of two positive definite matrices, $\mathrm{diag}(Q)$ and Q, and a positive semidefinite one, $y y^T$. It is hence positive definite and all of its eigenvalues are positive. Hence, there exists a positive constant c which satisfies the sufficient condition for the theorem to hold. □

Asynchronous update algorithm

A natural generalization of the parallel update algorithms presented is the asynchronous update scheme where only a random subset of processing units update their α values at a given time instance. Notice that the synchronous update scheme can be thought of as a limiting case of this more general version where all units participate in a random subset update. Define the set of units that update at a given instance n as $\mathcal{M}_u^{(n)}$ and the rest as $\mathcal{M}_{no}^{(n)}$, such that $\mathcal{M}_u^{(n)} \cup \mathcal{M}_{no}^{(n)} = \mathcal{M}\ \forall n$. Then, the update algorithm for the i-th unit is:

$$\alpha_d(n+1) = \begin{cases} [\alpha_d(n) + \kappa_d G_d(\alpha)]^+ , \forall d \in S_i, \text{ if } i \in \mathcal{M}_u^{(n)} \\ \alpha_d(n), \forall d \in S_i, \text{ if } i \in \mathcal{M}_{no}^{(n)}, \end{cases} \tag{9.12}$$

where $G_d(\alpha)$ is defined in (9.10).

Asynchronous update schemes are in fact more relevant in practical implementations, since it is usually difficult for the units to synchronize their exact update instances. Two well-known conditions, which are given below, are together sufficient for the asynchronous convergence of a nonlinear iterative mapping $\mathbf{x}(n+1) = T(\mathbf{x})$ [38, p. 431].

Definition 9.5 (*Synchronous convergence condition*) *For a sequence of nonempty sets $\{X(k)\}$ with $\ldots \subset X(k+1) \subset X(k) \subset \ldots X$, we have $T(\mathbf{x}) \in X(k+1)$, $\forall k$, and $\mathbf{x} \in X(k)$. Furthermore, if $\{y^k\}$ is a sequence such that $y^k \in X(k)$ for every k, then every limit point of $\{y^k\}$ is a fixed point of T.*

Definition 9.6 (*Box condition*) *Given a closed and bounded set X in $\mathbb{R}$, for every k, there exist sets $X_i(k) \subset X$ such that*

$$X(k) := X_1(k) \times X_2(k) \times \cdots \times X_M(k).$$

For the gradient projection algorithm (9.10), it is straightforward to apply the results of the convergence analysis above to the asynchronous update case. Towards this end, define the sequence of nonempty, convex, and compact sets

$$X(k) := X_1 \times X_2 \times \cdots X_M,$$

where $X_i := [x_i^* - \delta(k), x_i^* + \delta(k)]\ \forall i$ and $\delta(k) := \|\alpha(k) - \alpha^*\|$. Since $\delta(k+1) < \delta(k)$ by Theorem 9.4, we obtain

$$\cdots \subset X(k+1) \subset X(k) \subset \cdots X.$$

Here, X is defined as the interval $[0, \alpha_{max}]$ where a sufficiently large fixed α_{max} is chosen without any loss of generality due to the existence of a finite equilibrium point and geometric convergence. Hence, the box condition is satisfied by the definition of $X(k)$. Since $\delta(k)$ is monotonically decreasing in k by Theorem 9.4, the synchronous convergence condition also holds. Therefore, the next convergence result for the asynchronous counterpart of the parallel update algorithm (9.10) immediately follows from the asynchronous convergence theorem [38, p. 431]:

Theorem 9.7 *Assume that the conditions in Theorem 9.3 hold. If a random subset of the units update their* α *values at each iteration according to (9.10) while others keep theirs fixed, then the resulting (totally) asynchronous update algorithm defined in (9.12) converges to the unique maximum* α^* *of (9.4).*

Stochastic update algorithm

All of the update schemes described heretofore require an information exchange system (see Remark 9.1) to function properly. Also, the information flow within the system has been assumed to be perfect, i.e. the units have access to all the parameters needed by the update algorithms. However, this may not be the case in practice for a variety of reasons such as **communication errors**, i.e. if the units are not collocated or approximations are imposed in order to reduce the communication load between the units. To accommodate this, consider the update algorithm

$$\alpha(n+1) = [\alpha(n) + \kappa s(n)]^+ . \tag{9.13}$$

Define $s(n) := G(\alpha, n) + \beta(n)$, where $G(\alpha, n)$ is defined componentwise in (9.10). Here, $\beta(n)$ is the random *distortion* at time instance n. It is possible to establish convergence of the algorithm (9.13) which can be interpreted as a parallel update scheme under **stochastic distortions**, if these random perturbations satisfy some conditions. Toward this end, let us characterize the relationship between the distortion term $\beta(n)$ and the real gradient $G(\alpha, n)$ through the variables $\rho(n) > 0$ and $-\pi \leq \theta(n) \leq \pi$:

$$|\beta(n)| = \rho(n)\,|G(\alpha, n)|, \quad \cos(\theta(n)) = \frac{\beta^T(n) G(\alpha, n)}{|\beta(n)|\,|G(\alpha, n)|} . \tag{9.14}$$

Theorem 9.8 *Let the stochastic distortion* $\beta(n)$ *defined in (9.14) through parameters* $\rho(n) > 0$ *and* $-\pi \leq \theta(n) \leq \pi$ *satisfy:*

$$\rho^2(n) + 2\rho(n)\cos(\theta(n)) + 1 > 0, \forall n,$$

and

$$\frac{1 + \rho(n)\cos(\theta(n))}{(1 + \rho(n))^2} \geq \bar{K} > 0, \forall n,$$

where $\bar{K}$ *is a positive real number. Then, the update algorithm (9.13) converges to the unique maximum,* α^**, of the objective function* F *in (9.4) geometrically, if the elements of the step-size vector,* κ_d*, satisfy*

$$0 < \kappa_d < \frac{2\bar{K}}{\sqrt{\max \lambda(A^T A)}}, \forall d.$$

Proof The proof makes use of the upper-boundedness and strict concavity of the function F and Lipschitz continuity of its gradient $G(\alpha) = \nabla F$ as in the proof of Theorem 9.3. Modify the function F, without any loss of generality, such that it is bounded above by zero. It is straightforward to show that the conditions in the theorem on the distortion, $\beta(n)$, imposed through parameters $\rho(n)$ and $\theta(n)$ ensure that

$$\|s(n)\|_2 \geq k\,\|G(\alpha, n)\| \;\forall n,$$

where $k > 0$ is a positive constant and

$$s(n)^T G(\alpha, n) \geq \bar{K}\,\|s(n)\|_2^2 \;\forall n.$$

These conditions, together with that on the positive step-size constant vector k, ensure that the update algorithm (9.13) converges to the unique maximum of the objective function F, which follows immediately from Propositions 2.1 and 2.3 in reference [38, pp. 204–206]. Furthermore, since the objective function is strictly concave, the rate of convergence is geometrical (see Proposition 2.4 in reference [38] for details). $\square$

The conditions in Theorem 9.8 are now investigated through a couple of **illustrative examples**.

Example 1
Let $-\pi/2 < \theta(n) < \pi/2 \;\forall n$. The theorem holds for any value of ρ as long as the condition on the step-size κ is satisfied, where $\bar{K} \leq (1+\rho(n)\cos(\theta(n)))/(1+\rho(n))^2 \;\forall n$. This means in practice that if the angle between the gradient $G(\alpha)$ and distortion β vectors remains less than 90°, then it is possible to choose a sufficiently small step-size to compensate for the errors and ensure convergence.

Example 2
Let $|\theta(n)| > \pi/2 \;\forall n$. If $\rho(n) < 1$, then the step-size can be chosen sufficiently small and the conditions in the theorem can be satisfied. In this case, the distortion is in the almost opposite direction of the correct gradient, and hence its norm with respect to the gradient needs to be bounded.

9.3.3 Active set method and a numerical example

Although the update algorithms presented in Section 9.3.2 provably converge to the unique solution of (9.4), and hence approximately solve the original binary classification problem (9.1), they often result in a large number of support vectors. This is undesirable not only for efficiency reasons but also due to the communication overhead it brings to the system. To address this problem, *active set* methods are proposed, which have been widely used in solving general quadratic problems. Furthermore, they have also been applied to SVMs in centralized classification formulations [122, 158, 187].

Active set algorithms solve the problem iteratively by changing the set of active constraints, i.e. the inequality constraints that are fulfilled with equality, starting with an initial active set $\mathcal{A}_0$. Since the initial set is in most cases not the correct one, it is modified iteratively by adding and removing constraints according to some criteria and testing if the solution remains feasible [187]. Active set methods are considered to be more robust and better suited for warm starts, i.e. when there is some prior knowledge on the initial set [94]. For the relaxed problem (9.4), there are only non-negativity constraints on α. In this case, the vectors with $\alpha > 0$ constitute the set of SVs. Hence, the set of SVs is the complement of the active set and both sets are mutually exclusive. Furthermore, the union of both sets gives the (universal) set of all feature vectors.

A combinatorial trial-and-error approach is infeasible in practice except from very small constraint sets. Several schemes have been proposed in the literature for **iteratively updating the active set**. However, popular approaches such as *line search* [158] are not suitable for distributed implementation. The selection of active constraint sets has also been studied in the context of the simplex method [106]. In order to minimize the communication overhead and simplify implementations, a greedy approach has been proposed [11] for updating the active set, similar to those in the literature. For many problems, such heuristic methods are known to perform well [162].

The **greedy algorithm** adopted for updating the active set is summarized as follows. At each iteration, the data point with the highest positive gradient (derivative of the objective function F in (9.4)) is removed from the active set as an SV candidate. Next, the problem described in Section 9.3.2 is solved through parallel updates under the resulting active set constraints. Finally, the SV with the lowest α is added to the active set which is a rough approximation to finding the constraint that is violated by the solution.

The last step in the algorithm both speeds up the convergence and ensures a **user-defined upper-bound** on the number of SVs, thus turning it into a tunable parameter of the scheme. The total number of active set updates (iterations) in the algorithm should be at most as many as the desired number of SVs in general unless specific assumptions are made on the initial active set. Due to its iterative nature, this algorithm is also very suitable for online learning where training data is dynamic. Clearly, there is no need to rerun the whole algorithm in order to incorporate new data into the decision function each step.

Communication overhead

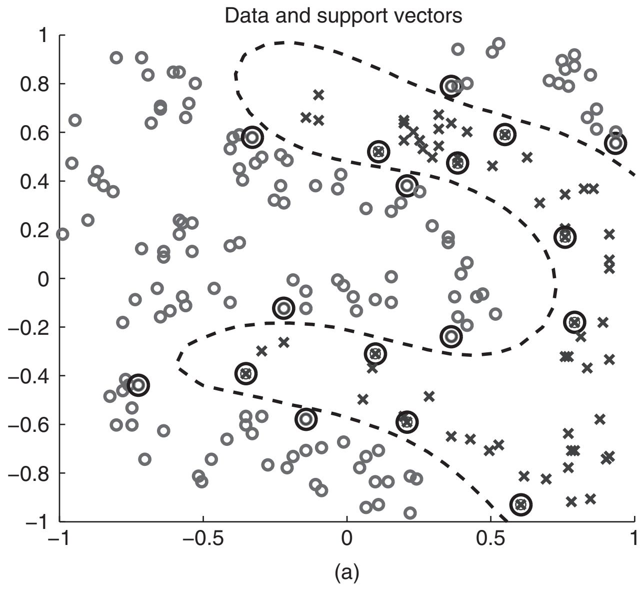
The communication overhead of the active set-based approach is significantly lower than the plain algorithm presented in Section 9.3.2. During parallel updates at each active set stage (inner loop), individual nodes exchange only the scalar α parameter values. They send an SV to the system node only when there is a change in the active set. Naturally, the amount of information sent by each unit would have been excessive if there were no active set restriction and all the corresponding data points were transmitted. Therefore, limiting the number of SVs through the active set decreases the communication load significantly. The information flow within the distributed system results, thus, in an efficient communication scheme.

Numerical example

The framework presented is analyzed on a synthetic but complex two-dimensional[4] dataset and compared with the centralized solution. First, the problem is solved centrally using a standard radial-basis-function (RBF) kernel and quadratic programming tools

[4] This two-dimensional dataset [76] is selected to facilitate visualization, and not because of any restriction of the algorithm on data dimension.

provided by Matlab. Next, the results of the distributed algorithm are obtained with an imposed upper-bound on the number of support vectors of 40, respectively. The starting point of the algorithm is random and perfect information flow is assumed in the system. The SVs of the centralized classifier and an example solution of the distributed one are depicted in Figure 9.2. It is observed that the distributed algorithm performs almost as well as the centralized one. The increased number of SVs needed for classification and



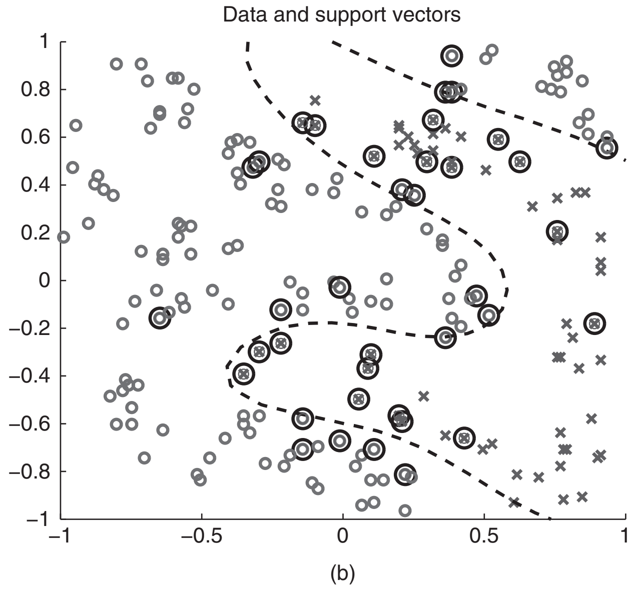


Figure 9.2 Comparison of (a) a centralized classifier with (b) the distributed one.

slight performance degradation in the distributed case are attributed to the approximate nature of the algorithm and the specific penalty function used in the relaxation of the problem. However, the presented framework, especially the active-set algorithm, has a lot of room for improvement in terms of, for example, preprocessing, better search heuristics, and post clustering of obtained SVs.

9.3.4 Behavioral malware detection for mobile devices

Widespread use and general-purpose computing capabilities of next generation smartphones make them the next big targets of malware and security attacks. Given the battery, computing power, and bandwidth limitations inherent in mobile devices, detection of malware on them is a nontrivial research challenge that requires a different approach from those used for desktop or laptop computing. The distributed machine learning and SVM classification framework presented has many suitable characteristics and can be applied to this problem as an alternative or additional defense method.

Mobile device usage patterns such as the number of SMSs sent and call durations can be exploited to collaboratively derive flexible, personalized, and behavioral signatures of malware. For example, a security laboratory can provide the malware behavior data while the participating users join the system with their normal usage data. Once a binary classification function (either specific to malware or for aggregate anomaly detection) is collectively trained, it is used to detect malware and other attacks. The distributed learning approach adopted provides multiple advantages:

1. It does not require the mobiles to send all of their behavior data to a security center. Hence, it is lightweight in terms of bandwidth usage.
2. Due to the abstract nature of feature vectors and not requiring a central repository for all of the user data, it preserves the privacy of the participating users.
3. The classification function learned is essentially an automatically generated behavioral (malware) signature that takes into account the usage patterns of ordinary users.

Given its favorable properties, this scheme provides a promising and low-overhead defensive layer for mobile devices, possibly alongside existing approaches.

Simulations and example implementation

A proof-of-concept illustrative prototype (Figure 9.3) is developed on smartphones with Symbian S60 operating system that communicates to the main server over HTTP. Both the server and clients use Python programming language [180] for ease of implementation. In addition, multiple clients are faithfully emulated on a single computer as part of more extensive simulations thanks to the interpreted nature of the Python language. Hence, the experimental infrastructure requirements are kept at minimum.

A dataset from the MIT reality mining project [60] is used for the simulations. It consists of phone call, SMS, and data communication logs collected via a special application during normal daily usage of volunteers. This usage data is preprocessed to generate histograms with a set of 20 features (see Table 9.2) over 6 h periods. Here,

Table 9.2 Histogram features

Feature (in numbers per 6 h intervals)
Short-duration calls (less than 2 min)
Medium-duration calls (between 2 and 6 min)
Long-duration calls (more than 6 min)
Short intervals between calls (less than 1 h)
Medium-length intervals between calls (between 1 and 3 h)
Long-length intervals between calls (more than 3 h)
Outgoing SMS
Short periods between outgoing SMS
Medium periods between outgoing SMS
Long periods between outgoing SMS
Incoming SMS
Short periods between incoming SMS
Medium periods between incoming SMS
Long periods between incoming SMS
Short-duration packet-sending activities
Medium-duration sending activities
Long-duration sending activities
Short periods between sending activities
Medium periods between sending activities
Long periods between sending activities

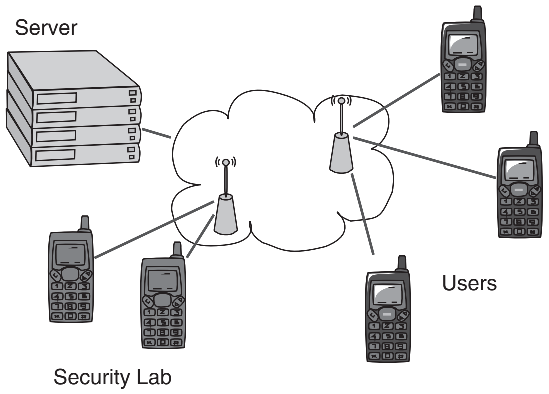


Figure 9.3 Illustration of the distributed malware detection system for smartphones.

the short periods or intervals refer to less than 1 h, medium ones to between 1 and 3 h, and long ones to more than 3 h, respectively. Short-call duration refers to less than 2 min, medium one to between 2 and 6 min, and long one to more than 6 min.

The selected example feature set is clearly statistical in nature, and hence, privacy preserving. The system focuses on aggregate and high-level usage characteristics rather than private data such as lists of people, time, and content. Thus, it is possible to run the algorithm on the server side without intruding into the privacy of the mobile user.

In one experiment, usage data from a subject over 50 days (200 samples) constitutes the training set. A malware is artificially injected into a test set of 25 days. The simulated malware behavior is based on well-known *Viver*[5] and *Beselo*[6] Trojans. These Trojans (malware) exhibit themselves by excessive SMS usage, e.g. sending out up to 20 SMSs in a short duration of time. The results obtained are promising and indicate high accuracy rates typically on the order of 90 percent or more.

9.4 Discussion and further reading

A comprehensive overview of intrusion detection and prevention systems can be found in references [25, 26]. The thesis, reference [191], presents use of internal and virtual sensors for intrusion detection. The base-rate fallacy is defined and discussed in detail in reference [24].

References [40, 162] provide an excellent introduction to machine learning and the kernel methods mentioned in Section 9.2. A recent book focusing on applications of machine learning to security is reference [103].

The distributed machine learning framework in Section 9.3 is based upon references [11, 12]. It has been recently applied to behavioral malware detection for mobile devices (smartphones) as discussed in Section 9.3.4. Another recent application of machine learning (probabilistic diffusion) to malware detection on mobile devices is reported in reference [13], which also contains an overview of the related literature in this area.

[5] http://www.f-secure.com/v-descs/trojan_symbos_viver_a.shtml
[6] http://www.f-secure.com/v-descs/worm_symbos_beselo.shtml