KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Privacy-friendly machine learning algorithms for intrusion detection systems

Henri De Plaen

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: wiskundige ingenieurstechnieken

Promotor:
Prof. Dr. ir. Bart Preneel

Assessoren:
Prof. Dr. ir. Karl Meerbergen
Prof. Dr. ir. Sabine Van Huffel

Begeleiders:
Dr. Aysajan Abidin
Dr. Abdelrahaman Aly

Academiejaar 2018 – 2019

Preface

I would like to thank everybody who kept has been involved in the production of this work, my promotor for his patience and especially my supervisors for the long discussions, insight, feedback and availability. I would also like to thank the jury for reading the text. My sincere gratitude certainly goes to my family and my girlfriend who supported me.

Henri De Plaen

Abstract

In this thesis, we present a set of practically usable machine-learning algorithms for intrusion detection systems using multi-party computation (secret sharing). This allows a user to query an already trained model preserving both the privacy of the query and of the model. We investigate two major classes of machine-learning algorithms broadly used on anomaly-based intrusion detection systems: support vector machines, both linear and non-linear, and nearest neighbors classifiers. In addition, different data-reduction methods in the feature space are investigated such as the principal components analysis decomposition or the chi-square feature selection. Instance space reduction methods are also investigated such as the condensed nearest neighbors algorithm, which has been applied for the first time to intrusion detection systems, or the k-means clustering. We also systematically compare two different multi-class models for support vector machines.

Based on these algorithms, we investigate how they can be made privacy-friendly. Different methods to achieve the privacy-friendliness are briefly described such as differential privacy, fully homomorphic encryption, garbled circuits and secret sharing. We justify our choice for the secret sharing and explain how we can use it to achieve a privacy-friendly evaluation on the classifiers. Finally, we benchmark the results of the privacy-friendly algorithms and their variants using data reduction.

Linear support vector machines allow a rapid evaluation for a good accuracy. The best performance is achieved using the chi-square reduction. Higher accuracies can be achieved with non-linear support vector machines and nearest neighbors. However, compared to nearest neighbors, non-linear support vector machines are much more expensive using multi-party computation due to the need for dual evaluation. Nearest neighbors are also very expensive, but can be reduced to practically feasible evaluation times using the condensed nearest neighbors beforehand. This way we exploit the trade-off between expensive clear pre-processing and a lightweight secret model. When applying feature size reduction to the nearest neighbors, the PCA reduction seems more adapted than the chi-square feature selection.

Samenvatting

In dit masterproef presenteren we een set van praktisch bruikbare algoritmes voor inbraakdetectiesystemen die gebruik maken van multi-party computation (secret sharing). Dit laat een gebruiker toe om een reeds getraind model te bevragen met behoud van zowel de privacy van de query als de modelparameters. We onderzoeken twee belangrijke klassen van machine-learning algoritmes die op grote schaal worden gebruikt op anomaliegebaseerde inbraakdetectiesystemen: support vector machines classificatoren, zowel lineaire als niet-lineaire, en naaste buren classificatoren. Daarnaast worden verschillende datareductiemethoden in de feature dimensie onderzocht, zoals de principal component analysis decompositie of de chi-kwadraatselectie. Reductiemethoden in de instance set worden ook onderzocht, zoals het gecondenseerde naaste buren, dat voor het eerst wordt toegepast op inbraakdetectiesystemen, of de k-means clusteringmethode. We vergelijken ook systematisch twee verschillende multi-class modellen voor ondersteunende vectormachines.

Op basis van deze algoritmes, onderzoeken we hoe ze privacyvriendelijk kunnen worden gemaakt. Verschillende methodes om de privacy-vriendelijkheid te bereiken worden kort beschreven zoals differentiële privacy, fully homomorphic encryptie, garbled circuits en secret sharing. We rechtvaardigen onze keuze voor het gebruik van secret sharing en leggen uit hoe we deze om tot een privacyvriendelijke evaluatie van de classifiers te komen. Tot slot benchmarken we de resultaten van de privacyvriendelijke algoritmes en hun varianten die gebruik maken van datareductie.

Lineaire support vector machines maken een snelle evaluatie mogelijk voor een goede nauwkeurigheid. De beste resultaten worden bereikt met de chi-kwadraatselectie. Hogere nauwkeurigheid kan worden bereikt met niet-lineaire support vector machines en de naaste buren. In vergelijking met de naaste buren zijn niet-lineaire support vector machines veel duurder met multi-party computation omwille van de nood voor een duale evaluatie. Naaste buren zijn ook duur, maar kunnen worden gereduceerd tot praktisch haalbare evaluatietijden met behulp van de gecondenseerde naaste buren. Op deze manier maken we gebruik van de trade-off tussen dure clear pre-processing en een licht geheim model. Bij de toepassing van de feature reductie bij de naaste buren lijkt de PCA-reductie meer aangepast dan de chi-kwadraatfunctieselectie.

Contents

List of Figures

List of Tables

List of Algorithms

List of Symbols

Symbols

General symbols

x_i	feature vector;
y_i	corresponding class;
$\mathcal{E}_k(\cdot)$	encryption function (with eventual key k provided);
$H(\cdot)$	hash function;
$\mathcal{O}(\cdot)$	computational complexity;
$x\|y$	concatenation of x and y;
n_{svm}	number of support vector machines;
n_{sv}	number of support vectors;
n_q	number of queries;
n_{pca}	number of components kept after PCA transformation;
n_c	number of attack classes;
n_f	number of features (feature vector dimension);
n_t	number of training instances (training set size);
n_{χ^2}	number of selected features with the χ^2-measure;
k	number of neighbors in the k-NN algorithm;
t_p	number of true positives (binary classification);
t_n	number of true negatives (binary classification);
f_p	number of false positives (binary classification);
f_n	number of false negatives (binary classification).

Symbols specific to chapter 3

$\langle \bar{a} \rangle$	secret integer;
$\langle \bar{a}_i \rangle$	share of secret integer $\langle \bar{a} \rangle$;
$\langle \tilde{a} \rangle$	secret fixed point number;
$\langle A \rangle$	vector or matrix of secret numbers (integer or fixed point);
$\langle a \rangle_i$	secret number in vector $\langle A \rangle$ (integer or fixed point);
$\langle a \rangle_{ij}$	secret number in matrix $\langle A \rangle$ (integer or fixed points);
$\langle A \rangle_{i:}$	vector corresponding to the i-th line of matrix $\langle A \rangle$ (integer or fixed point);
$\langle A \rangle_{:j}$	vector corresponding to the j-th column of matrix $\langle A \rangle$ (integer or fixed point);
$\mathbb{Z}_{\langle k \rangle}$	space of secret integers of size k;
$\mathbb{Q}_{\langle k,f \rangle}$	space of secret fixed point numbers of discriminand size k and exponent f.

List of Abbreviations

Abbreviations

IDS	Intrusion detection system
MPC	Multi-party computation
PCA	Principal component analysis
KPCA	Kernel principal component analysis
SVM	Support vector machine
LSVM	Linear support vector machine
RBFSVM	Support vector machine with radial basis kernel function
k-NN	Nearest neighbors with k neighbors
CNN	Condensed nearest neighbors
O-A-A	One-against-all (multi-class model)

Chapter

1 Introduction

More than ever, information today is equal to money and power. The four most valued companies in the world (Microsoft, Apple, Google and Amazon) all have a significant part of their business model based on information and data. Furthermore, the rest of the industry has also fully entered into this information mutation: data is now everywhere. And so is sensitive data such as medical, governmental or industrial data. This characterizes the new information age we now live in. However, networks are now experiencing more and more attacks of various types trying to recover this information. The new information paradigm has to come with its new defense mechanisms.

Machine-learning has allowed improvement of many models, including the ones identifying network attacks. Though, these models have to be constructed using significant amounts of data, which are often based on previous real attacks making them sensitive. These models also ideally should be used as much as possible to better protect the information. We thus face the following paradox: sensitive data should be used to improve the protection of sensitive data.

This paradox can be solved by using multi-party computation (MPC). This allows for different parties to commonly compute an agreed-upon function where its input remains private and the output is revealed to certain parties. This solution allows a user to query a machine-learning algorithm trained on external data-bases, for its own defense while maintaining the privacy of all the data: the query and the data-base, on which the machine-learning algorithm has been trained. In other words, these solutions allow the use of sensitive data for a certain purpose without revealing it. This is a specific case of the now trendy *private-data as a service* (PDaaS).

However, if MPC allows us to solve our paradox, it also has a huge drawback: every operation is proportionally much more costly. We can thus not use MPC as we would use classical algorithms and must actively investigate the trade-offs we can make to reduce these costs. This thesis investigates how machine-learning can be used to evaluate a query based on an external data-base while preserving the privacy of the data, in the specific case of *intrusion detection systems.*

1.1 Related works

Intrusion detection systems are network security techniques that aim to detect malicious activity by comparing it to an existing data-base based on previous malicious activities [6]. The general philosophy is that an intruder has different behaviour than a legitimate user. It can be based on two methods [90]. Rule-based, which is also known as *signature-based*, aims to compare a new query to a data-base of specific patterns — the so-called signatures — that contain the sole reduced information to detect an anomaly. The big advantage of these methods is that they are very lightweight and can easily detect attacks that correspond to these signatures. Their disadvantage however is their very strong dependence on these signatures and the inability to detect anomalies that are not contained in the signatures. These signatures are furthermore often specific to certain systems or architectures [43]. In other words, signature specific methods almost always detect the same anomalies as the ones it has in its signature data-base, but fails at detecting anything that diverges from it [49]. This problem is solved by an alternative to signature-based methods: *anomaly-based methods*. They rely on more abstract, statistical models that are able to generalize the attack patterns and by this way detect attacks that have not been previously identified [26]. However, the generalization property has also its drawback as they tend to also over-identify normal activities as anomalies, resulting in a high false positive rate and a low false negative rate [55]. A lot of research has been conducted on these issues in the last 20 years, leading to very effective anomaly-based systems, by replacing more simple statistical models by more complex machine-learning algorithms [87]. The use of intrusion detection system is more and more critical in our connected world and is now widely used in the industry due to the potential high financial cost of failing to detect anomalies [13], [70]. Although signature-based systems are still used by some industry majors such as Cisco Systems, the wide majority is now consisted of anomaly-based models [73].

A big improvement in the last years is the use of distributed intrusion detection systems to monitor data across multiple nodes. Yegneswaran et al. have built the DOMINO overlay system, which proves to be able to detect attacks that could not be detected on isolated nodes and furthermore increases the general detection speed [98]. The general philosophy behind distributed intrusion detection systems is that users are not fixed, they change their ID and their target [80]. In a certain way, distributed intrusion detection systems gain more efficiency through the correlation of different individual data-bases [72]. The main problem when working with different data-bases is the privacy of the data. Information about the network use of a user cannot be made available to the public, nor in the distributed intrusion detection system itself, especially when it is distributed among different and independent networks. The processing of these data-bases through the system's algorithms has to be made privacy-friendly.

There are different approaches to privacy-preserving data-mining [58]. The first one is based on differential privacy, where the results are still treated in clear but in anonymized statistical manner to limit a maximum the possibility of de-anonmyzing the data-points [34]. This approach has been applied to various machine learning algorithms such as support vector machines or image processing [64], [99].

The other approach is to use cryptographically secure algorithms where the data is cryptographically hidden and processed in its hidden form to be finally revealed to the original data-owner. The big advantage of this method is that there is no possible way of finding statistical information about the data processed, which is not the case of differential privacy. However, these advantages have a cost: they are computationally very expensive algorithms. This is called homomorphic encryption. An alternative is to use *multi-party computation* where the data is distributed between different players which then process the information in such a way that the information cannot be revealed if sufficient participants are staying honest. These methods are based on ideas developed by Yao and Rabin or secret sharing [95], [65] and [76].

Multiple algorithms have been implemented using multi-party computation for aggregation and statistical analysis of network events such as SEPIA [19]. However, it has the drawback of being limited to two parties and only simple aggregation operations and basic statistical measures. Bogdanov et al. are proposing Rmind, a set of similar cryptographically secure toolboxes, this time with more than one party, but also limited to simple statistical algorithms, e.g. linear regression [16]. More complex machine-learning algorithms are still timid.

Cryptographically private support vector machines have been theorized by Laur et al. [48], but the only real implementations known to us are given in the domain of image-processing using linear kernels [50]. Barnett et al. are using polynomial kernels, but using homomorphic encryption and not multi-party computation, still in the domain of image classification [10]. There are also some implementations using neural networks, always residing on homomorphic encryption [33].

Shaneck et al. have been the first to propose privacy-friendly nearest neighbors using multi-party computation, but with a clear query point [77]. Methods encrypting both the query and the data-base have been implemented using homomorphic encryption [41], [91], but appear not to be secure over plaintext attack [96]. Qi et al. have proposed a specific two-party protocol based on additive secret sharing, without any implementation [63].

To our knowledge, no privacy-friendly machine-learning algorithms for intrusion detection systems have been implemented using multi-party computation and not much has been done in privacy-friendly machine-learning algorithms using MPC in general.

In general, intrusion detection systems are considered to be classification problems. However, the classifiers cannot use the raw data as such. Before being used in a machine-learning, the packets — which are of huge size — have to be reduced to a set of features [90] which is usually quite big independently of the method chosen to generate them [24], [75] [59]. A lot of different machine-learning algorithms have been tested and used on intrusion detecting systems [86]. The first ones are neural networks which tend to be very effective for binary classification but less at multi-class [3], [38], [56]. A second type of algorithms are decision trees [53], [61] and more generally random forest classifiers [81]. However, decision trees and random forests are rule-based methods which are much more difficult to hide and to make privacy-friendly. Multi-party computation is difficultly compatible with rule-based methods. Fuzzy logic [71], [79] is also used but much less common. Another technique is naive Bayes [22], but tends to have a lesser accuracy, and hidden Markov models [23], [88] that suffer from the same problem. The last two and most used methods are support vector machines and nearest neighbours methods. A lot of less classical methods have also been tested, but little research has been focused on them [74].

Mukkamala et al. have successfully used support vector machines to make the distinction between normal and attack classes using a simple SVM with Gaussian kernels [56]. Much more complex models have also been used combining KPCA and SVM and training the parameters with genetic algorithms [47]. Ming et al. have showed that adding data-points to the data-set based on aggregation of that same data-base allows to discover more complicated attacks with support vector machines [54]. This same idea has also been successfully implemented using random forests classifiers [3]. Investigations have also been conducted on how to reduce the training set size of SVMs for intrusion detection systems training [46], [97]. In general SVMs are able to detect almost all kinds of problems if the model it is used in is well built. This shows the importance of the research around support vector machines for intrusion detection systems.

Due to the cost of multi-party computation, the data-set sizes used during the secure computation of the algorithm have to be reduced as much as possible without losing accuracy. The feature-size reduction has been implemented using principal component analysis decomposition and χ^2 feature selection [35], [42], [83], [93]. In general, it seems that SVM are almost always using Gaussian kernel functions, sometimes polynomial, but never linear.

The nearest neighbours algorithm has also been studied for feature reduction with PCA and KPCA [36]. However, algorithms for reducing the data-points number like the one proposed in [7] seem to never have been tested.

1.2 Claims

In this thesis, we describe and analyze a way for a user to query an external machine-learning algorithm for the classification of a possible attack (figure 1.1). More specifically, The machine-learning model has been trained beforehand and is described my model parameters. Both the privacy of the query and the model parameters are preserved trough multi-party computation; the type of machine-learning algorithm queried is known as well as the reductions used. As the execution of MPC-based algorithms is very expensive, we suggest outsourcing (in a cloud) it to a group of servers (in our case 3), but the protocols described can directly be used between the different parties without outsourcing, in any n-party setting. All the interactions between the cloud, the user and model owners(s) are done using secure connections (TCP), as well as the connections between the servers of the cloud.

We provide algorithmic solutions to the evaluation of support vector machines using secret sharing. Both linear as radial-based functions are implemented. To the best of our knowledge, this is the first time non-linear support vector machines are implemented under MPC security constraints. Due to the cost of MPC, the training of SVMs becomes very expensive. These algorithms are thus limited to evaluation against already trained models and thus the use of only one model owner. We showed that

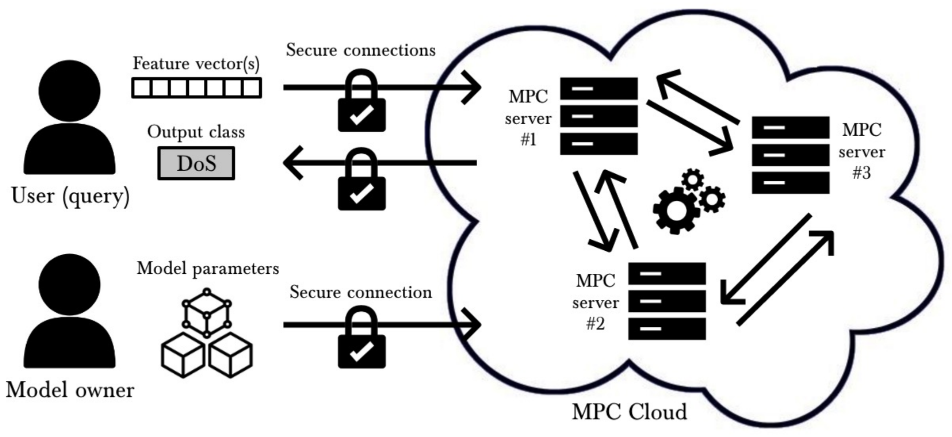


FIGURE 1.1: Proposed setting for privacy-friendly machine-learning algorithms for intrusion detection systems.

linear SVMs allow rapid evaluation of a query, especially when reduction methods are used. Non-linear SVMs are too slow, even with the reductions investigated.

We also provide algorithmic solutions to the nearest neighbors evaluation using the same secret sharing. We show that condensed nearest neighbors allow to make nearest neighbors practically feasible in the case of privacy-friendly intrusion detection systems.

Trade-offs that can be made to reduce the various costs of classical support vector machines and nearest neighbors, are investigated. Various reduction methods are pushed to their limits to see if they allow both of these algorithm to run in reasonable times to be used with multi-party computation. This comprises research in both the domains of machine-learning algorithms for intrusion detection systems and MPC-based machine-learning algorithms.

Furthermore, this thesis systematically compares the implementation of two different SVMs multi-class models: one-against-all and tree-based models. Up to now, they were alternatively used in the literature without systematic comparison. Tree-based models appear to be much faster without a loss of performance, both with linear as non-linear support vector machines.

We also investigate how feature size reduction methods such as PCA reduction or χ^2 feature selection impact the classification performance and speed of the algorithm. To our knowledge, χ^2 feature selection has never been applied before to nearest neighbors algorithms in the case of intrusion detection systems.

Next to investigating feature size reduction, we also investigate instance-set (or training set) size reduction: k-means and condensed nearest neighbors. To our knowledge, none of these algorithms have been applied to nearest neighbors in the case of intrusion detection systems. The condensed nearest neighbors allow to dramatically decrease the computation cost of the evaluation.

The two algorithms we present are practically usable in the proposed setting as they have limited computational, communication and round cost, for a totally satisfying classification accuracy.

1.3 Organization of this thesis

This thesis is organized in five main chapters, including this one.

- The second chapter presents the different reduction methods used as well as the SVM and the nearest neighbors models.
- The third chapter explains the existing techniques to achieve privacy and a justification for the method we opted for, the context and theoretical foundations used in the method we opted for as well as the algorithms we used using this setting.
- The fourth chapter presents the results for each model, first the reduction methods and then the results on MPC. The models are tested in this order: linear SVMs, non-linear SVMs and finally nearest neighbors.
- The fifth chapter finally concludes the thesis summarizing the results and presenting directions for future works.

A lot of data has been produced during the evaluation of the results of the different algorithms. We tried to limit ourselves to a strict minimum in the fourth section. Additional results can be found in the appendix. The appendices also contain information on the code used for the various implementations.

Chapter

2 Machine-learning algorithms for intrusion detection systems

2.1 Intrusion Detection Systems

Intrusion detection systems (IDS) are a brick in the existing defence algorithms arsenal wall of information security. More specifically, it includes a series of mechanisms that monitor network nodes and detect intrusions, i.e. malicious activity or policy violations. An IDS usually analyzes the incoming packets and tries to detect the suspect ones. In most cases, IDS are defined to be the sole surveillance application and do not comprise the control application: how the suspect packets are treated after a notification is not considered as being part of the IDS, the latter only focuses on the monitoring, analysis and notification [55]. Classically, the reports are made to an administrator or another competent entity, as *security information and event management* (SIEM) [13], which are then in charge of the control application.

IDS should not be confused with *firewalls*, but merely be seen as a complement of it. The only role of firewalls is to ensure that communication policies are followed carefully. A first difference is that firewalls have an upstream role whereas IDS are working downstream. In other words, firewalls are verifying that each packet is following carefully one of the pre-defined allowed communication protocols, before it enters the local network. An IDS analyzes the packets after they enter the local network to control if they show no abnormal behaviour. Another difference has already been mentioned: firewalls consider each packet separately whereas IDS can consider group of packets and thus look at a communication as a whole. In this sense, IDS are much more suited against *denial of service* (DoS) attacks than classical firewalls. A last difference concerns the exact scope of the packet analysis. As firewalls only have to enforce communication policies, they only have to look at the packet header, whereas IDS are searching for abnormal behaviours and are thus looking at the packets on the whole. To summarize this all, let's consider a high security building: the firewall would be equivalent to the agents allowing or not each individual to enter the site by carefully inspecting their papers, whereas the IDS would be the security agents monitoring the cameras inside to building searching for abnormal behaviour.

IDS should also not be confused with *anti-viruses* applications — though the term *anti-malware* would be more suited nowadays — that refer to the application layer in charge of the detection and control of malicious code, or malware. The first difference concerns the scope of their analysis: anti-viruses are analyzing (executable) code on a system more specifically than packets on a network. The second difference is similar as before: anti-viruses analyze code before it is allowed to be executed by the system and IDS analyze packets that already entered the network.

However, all these taxonomy classifications are to be considered with some flexibility. As the attacks become more and more sophisticated, security entities are incorporating more, and more subtleties are extending the scope of their detection methods. As such, they integrate other types of methods classically defined by other entities.

2.1.1 How IDS work

As briefly stated before, IDS have three main components: monitoring, analysis and notification.

The monitoring can be achieved in real time or at regular interval on different types of nodes, which define the type of IDS: network based IDS (NIDS), host based IDS (HIDS) and hybrid if they monitor on both.

The analysis is the core part of the IDS and is again divided in three main components: the extraction of features, the pattern analysis and the final classification [90]. This will be the part which will interest us in this thesis.

As written before, the notification is done to a controller, either an administrator or a SIEM. Classically, this takes place in the form of a series of logs which are later examined by the controller. In this sense, the speed is not the main focus of an IDS, but rather the correct identification of intrusions. However, one can also consider *intrusion prevention systems* (IPS) which are working upstream. The literature sometimes considers these systems to be a specific class of IDS or to be a category on their own. However in opposition to IDS, IPS need to be fast and thus usually use signature-based detection. In this sense, IPS can be seen as an extension of firewalls as they also analyze the content of packets and not just the enforcing of protocols. Taking the IPS into account, one should still consider analysis speed in IDS.

The general structure of IDS is summarized at figure 2.1.

2.1.2 Extraction of features

Packets to be analyzed have a huge size and cannot be analyzed as such by the IDS. They typically incorporate huge redundancy and other non-necessary information such as padding. The idea of feature reduction is to reduce the packets to a limited number of features, representing as much non-redundant information as possible. Examples of features extracted consist of the connection length, the protocol used, the number of bytes transferred from the source to the destination and vice-versa.

2.1.3 Pattern analysis

The goal of the analysis of the IDS is to categorize the packets into different classes, typically a normal class and some different attack classes. As stated in the introduction, this can be done by two different manners based on a signature (*signature-based*) and

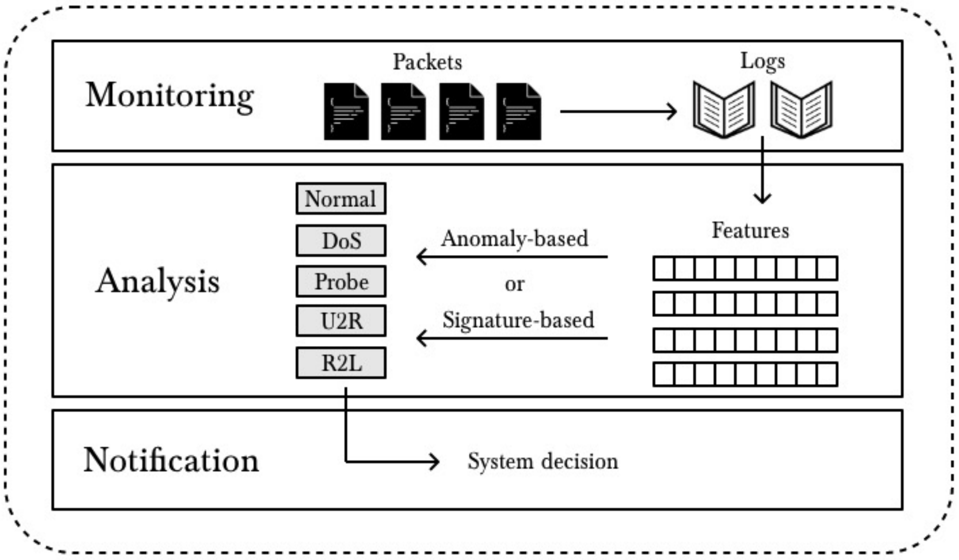


FIGURE 2.1: General structure of an intrusion detection system. The different attack classes are based here on chapter 4.

using some statistical rules (*anomaly-based*). In this thesis, we will interest us to anomaly-based IDS, more specifically using classification machine-learning algorithms. The goal is that a user is able to classify its (query) on external trained machine-learning algorithm from one or more other parties, called the model owner(s). Furthermore, these algorithms have to be privacy-friendly in the sense that nor the query, nor the information resulting from the training should be revealed. This chapter aims at describing the two main machine-learning algorithms used without any consideration of the privacy-friendliness that will be covered in chapter 3.

2.2 Data reduction

As we will see in chapter 3, making an algorithm privacy-friendly is proportionally more costly: every operation takes much more time and are underlyingly based on a whole bunch of other sub-operations. Algorithms that may run with an acceptable speed in normal circumstances may become too slow to be used in practice in their privacy-reserving form. The need for data reduction thus becomes crucial. The algorithms presented in this section aim at reducing the data as much as possible, to suppress redundancy or information that is not relevant to the performance of the classification algorithms. Every information that is not crucial and kept will participate to the algorithm and unnecessarily slow it down.

We investigate here two main classes of data reduction. Instance-set size reduction aims at reducing the number of points used in the training of the algorithm and often find themselves in the final information used for the evaluation of this algorithm against a new data-point (a query). Feature-size reduction aims at reducing the size of the feature vector used as an input in the algorithm. This can also drastically reduce the speed of the algorithm. As an example, most of the algorithms that we present work by calculating the euclidean distance between two feature vectors at their core. The evaluation of an euclidean distance has a complexity depending linearly on the size of the vector. Reducing the feature vector size by 2 could reduce the algorithm speed by the same factor if the computation of the distance takes place in every iteration for example.

2.2.1 Instance-set size reduction

Instance-based learning algorithms are often faced with the problem of deciding which instances to store for use during generalization. A general principle of machine-learning is that the variance of its prediction always theoretically decreases if the training data-set increases — under a few assumptions. In other words, this means that the more points are used to train the algorithm, the better it performs. This is not a very surprising property as machine-learning algorithms are in essence just complex statistical models, for which the notion of increasing the sample size to reduce the error is very common. Reducing the instance-set must be done in a clever way to be able to keep a similar — or at least not significantly worse — performance as with the whole data-set.

In practice, data can be classified into two kind of classes, attack and no attack, or normal. In a practical intrusion detection system, most of the traffic is normal and only a small proportion is attacking. We thus want to reduce the size of the normal data-set to make it of a similar size to the attack classes. Furthermore, the normal instances are very diversified and one may not just reduce the set randomly at the risk of missing relevant instances. One must thus find a more intelligent heuristic to decide which normal instances to keep.

How can we decide which instances to keep? The idea is to keep only the ones near the decision boundary. If we consider all the data-points in a hyper-space of the dimension of the feature vector, the decision boundary is the set of hyper-surfaces separating the different classes. Indeed, if we classify all the points contained in the hyper-space — more accurately called varieties —, each one will be attributed to a certain class and points of the same class will then form regions in this hyper-space. The decision boundary is the boundaries of these regions, the varieties separating these regions. The idea behind the selection of the points near the decision boundaries is that points far from them have a small impact on them. A data-point that is very different from another one will have no impact on its classification. In other words, points have only a local impact in the hyper-space. This idea is at the core of a lot of machine-learning algorithms.

k-means clustering

The k-means clustering algorithm is a semi-supervised instance set reduction algorithm. This a specific case of clustering methods, that aim to divide the original data-set into smaller k smaller clusters that should share some common properties. The challenge here is to find the most homogeneous possible clusters, which instances are similar enough — referred to as the minimization of *intra-class inertia* —, but not too many so that the clusters are well differentiated – also referred to as the maximisation of *inter-class inertia.* After this being achieved, one can thus reduce each cluster far from the decision boundary to an archetype instance of this cluster and keep the clusters near the decision boundary. The algorithm is semi-supervised as the clustering in itself is only based on the features set, but is only applied on the data-points that are classified as normal, which of course depends on the target space.

To determine the different clusters, the k-means algorithm tries to minimize the distance between the instances in each cluster $\mathcal{S}_i$. The number of clusters is fixed to k, which an hyper-parameter of the model set by the user. Computing the distance between each of the points would require a quadratic at each evaluation of cluster. To keep the complexity linear, the distances are not computed between each of the instances, but between each of the instances and the mean of these instances $\mu_i = \left(\sum_{x_j \in \mathcal{S}_i} x_j\right) / |\mathcal{S}_i|$ where $|\mathcal{S}_i|$ is the number of instances in the cluster

$$\arg\min_{\mathcal{S}} \sum_{i=1}^{k} \sum_{x_i \in \mathcal{S}_i} \left\| x_j - \mu_i \right\|^2, \tag{2.1}$$

with the objective function $\sum_{i=1}^{k} \sum_{x_i \in \mathcal{S}_i} \left\| x_j - \mu_i \right\|^2$ being called the *within-cluster distance.*

Defining the optimal clusters would at least be exponentially complex in function of the number of total instances as each cluster combination would have to be tested. Therefore, the k-means algorithm starts with assigning the starting clusters to initial data-points, chosen at random in our case. It then goes through the whole data-set and assigns each point to the cluster with the nearest mean μ_i. The means are then updated and the process starts again. This is given at algorithm 1. Although this algorithm doesn't guarantee any optimality nor polynomial computational time, it is considered very effective in the practice [8].

```
Data: Feature data-set $\mathcal{F} = \{x_i\}_{i=1\dots n}$ and number of cluster sets $k$
Result: $k$ clusters $\mathcal{S}_i = \{x_i\}_{i=1\dots m_i}$
foreach $i = 1\dots k$ do  $\mu_i \leftarrow$ random $x_j$
repeat
    foreach $x_j \in \mathcal{F}$ do
        foreach $i = 1\dots k$ do $d_i \leftarrow \left\|x_j - \mu_i\right\|$
        $l \leftarrow \arg\min d_i$
        assign $x_j$ to $\mathcal{S}_l$
    end
    foreach $i = 1\dots k$ do
        $\mu_i \leftarrow \mu_i = \frac{1}{|\mathcal{S}_i|}\left(\sum_{x_j \in \mathcal{S}_i} x_j\right)$
    end
until convergence
return $\mathcal{S}_{1\dots k}$
```

Algorithm 1: The k-means algorithm. The convergence criterion typically is no more evolution in the means or the composition of the clusters, which is almost always equivalent in the practice.

Once the clusters are determined, we can remove the normal data instances that are the farthest from the decision boundary and replace the whole cluster by its mean μ_i. In this way, we reduce the number of points far from the decision boundary and keep all the ones near the decision boundary. The number of set to be reduced $p < k$ is also an hyper-parameter set by the user. A way to determine which one are the farthest is just to compute the distance between the mean of each cluster and the mean of the attack instances defined in the same way.

2.2.2 Feature size reduction

In the analogy of the hyper-space developed here-above, the feature-size was the dimension of the hyper-space. Reducing the feature size means working in a smaller hyper-space. This also means the ability of having less entangled boundaries to separate our classes. This suppresses complexity of the model, which is generally a good thing.

The main task in statistical learning is to avoid any *overfitting*: any useless addition of complexity may allow the regression to capture the variance of the specific data-set on which it is trained on top of the underlying relation is it supposed to capture. The regression would then reproduce the training set itself instead of generalizing it. I therefore would like to cite John von Neumann

> *With four parameters I can fit an elephant, and with five I can make him wiggle his trunk.*

In other words, one parameter is sufficient to add a lot of complexity to the model and increase a lot the risk of overfitting. The general rule of thumb is thus to always reduce as much as possible the complexity of the model and always stick to the lowest one able to reproduce the general scheme of the data. This principle is also known as *Occam's razor*.

However, we must reduce the feature-size in a clever way, using a mapping that takes into account the possible entanglement of our boundaries in the higher dimension hyper-space.

Principal components analysis

Let's consider a feature vector $x = \left[x^1, \ldots, x^{n_f}\right]^T$ where n_f is the number of features (the exponent notation is used to avoid confusion with the indices which represent the different feature vector instances). A linear Principal Components Analysis (PCA) dimensionality reduction consists in a spectral analysis of the covariance matrix defined by

$$\hat{\Sigma}_{ij} = \mathbb{E}\left[\left(x^i - \mathbb{E}[x^i]\right)\left(x^j - \mathbb{E}[x^j]\right)\right], \tag{2.2}$$

where $\mathbb{E}(x)$ represents statistical expected value of x. The covariance matrix is in fact a generalization of the variance to multiple dimensions and its elements actually measure the (linear) correlation between two variables and its diagonal the variances. This covariance matrix can diagonalized with nonnegative eigenvalues λ_i

$$\hat{\Sigma} x = \lambda_i x \tag{2.3}$$

The M greatest eigenvalues are then kept and the data are transformed accordingly:

$$z_i = x_i^T x. \tag{2.4}$$

Here is an attempt of an intuitive understanding of the process. Each input variable of the original input vector is independent, but some have strong underlying relations between them, measured by their covariance matrix. By diagonalizing it, there result n_f new variables without any linear underlying relation. In a certain way, the eigenvectors x_j represent these relations and their respective eigenvalues λ_j their contribution, thus their importance. We can then afford to drop the non-important ones.

In our case, each feature is first being normalized, i.e. zero mean and unit variance. In other words, the covariance matrix has a unit diagonal and with a feature dimension of n_f, we have $tr(\hat{\Sigma}) = n_f$ which corresponds in a certain way to the total variance of the whole feature space. Quite logically, the latter is invariant to any linear transformation as the trace is an invariant to any linear transformation. The main idea behind linear PCA analysis reduction is thus to keep the richness of the data to a maximum, thus its total variance, and reducing its dimension a maximum by dropping its smallest contributors. A good measure for the relative importance of each eigenvector, or underlying linear relation, is its fraction or variance contribution given by

$$f_j = \frac{\lambda_j}{\Sigma_{\forall j} \lambda_j} = \frac{\lambda_j}{\mathrm{tr}\left(\hat{\Sigma}\right)}. \tag{2.5}$$

One can then choose the input values it decides to keep in descending order of the respective fractions f_j.

Chi-square feature selection

Up to now, the idea was to reduce the feature size by constructing new ones as transformations (or mappings) of the original feature vectors. However, we can also choose to select a number of features and just not care about the other ones by simply discarding them. Another interesting approach would be to be able to reduce the feature size in a supervised way, taking the machine learning algorithm into account. These ideas can be tackled with χ^2 feature reduction.

This feature selection method is based on the χ^2 statistical hypothesis test that measures the dependence of two categorical variables. Two variables are independent if for all categories A and B

$$P(A,B) = P(A) \times P(B). \tag{2.6}$$

This value is observed to be N_{AB}, the frequency of respective categories for each variable A and B happening together. This is value is expected to be

$$E_{AB} = N_{A\cdot} \times N_{\cdot B}, \tag{2.7}$$

with $N_{A\cdot} = \sum_{\forall B} N_{AB}$ and respectively $N_{\cdot B} = \sum_{\forall A} N_{AB}$. Pearson showed in 1900 that the distance between these observed and the expected values follows a χ^2 distribution of degrees of freedom $DF = (c_A - 1)(c_B - 1)$ where c_A and c_B are the number of possible categories for A and B

$$\chi^2_{DF} \sim T = \sum_{\forall A,B} \frac{(N_{AB} - E_{AB})^2}{E_{AB}}. \tag{2.8}$$

In our case, the problem is that almost all of the input features are non-categorical and all of them are after pre-processing. A solution could be binning[1]. However, this would not allow us to do this while taking the classification algorithm into account and thus creating a selection tailored for the specific classification algorithm. This is why we define here a χ^2 metric to compare the output classification with the expected one. We will only work with two categories, true and false. This gives us the following metric

$$\begin{aligned} \chi^2\text{–metric} &= \frac{\left(t_p - (t_p + f_p)N_p\right)^2}{(t_p + f_p)N_p} + \frac{\left(tn - (f_n + t_n)N_p\right)^2}{(f_n + t_n)N_p} && (2.9) \\ &+ \frac{\left(f_p - (t_p + f_p)N_n\right)^2}{(t_p + f_p)N_n} + \frac{\left(t_n - (f_n + t_n)N_p\right)^2}{(f_n + t_n)N_p}, && (2.10) \end{aligned}$$

where t_p is the number of true positives, t_n the number of true negatives, f_p the number of false positives and f_n the number of false negatives.

Based on this metric, we can now now select the relevant features in an iterative manner. Each feature is temporary selected as being the only one in the feature vector and the resulting model is trained and tested through this metric. If the metric is

[1]This is a step of transforming the input data again into categorical data putting into different bins. This is a just a discretization into intervals of the distribution each feature.

smaller than a certain threshold, it is definitely deleted and otherwise restored. The process can then start again with a new feature until all have been tested. In this way, the algorithm is also independent of the order in which the features are tested. This allows to find an optimal set of features.

It has been observed that χ^2 feature selection performs very well against multi-class data-sets compared to other feature selection methods [93]. It has also successfully been applied to some cases of intrusion detection systems and seem to deliver better results than PCA [83].

2.3 Support Vector Machines

Support vector machines are a set of supervised machine learning algorithms than can be used either for classification or regression proposed by Vapnik in 1963 [89]. The main idea originates with binary classification and consists of finding an optimal hyper-plane between the data-points separating both target classes.

2.3.1 Linear support vector machines

In its primal form, the support vector machine corresponds to the following minimization problem

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \frac{1}{2} w^T w + C \sum_{k=1}^{N} \xi_k \\ \text{subject to} \quad & y_i \left[w \cdot x_i + b \right] \geq 1 - \xi_i, \; i = 1, \ldots, N \\ & \xi_i \geq 0, \; i = 1, \ldots, N. \end{aligned} \tag{2.11}$$

This corresponds to finding a hyper-plane defined by the vector w and intercept b that separates all the data-points. The minimization of $w^T w = \|w\|^2$ corresponds to maximizing the margin between the hyper-plane and the nearest data-points. This is the criterion used to define a unique hyper-plane as an infinity of potential hyper-planes that separates the data-points correctly may exist. Indeed, the nearest data-points x are on the parallel hyper-plane $w^T x - b = \pm 1$. The distance between the margin and the nearest point is thus $1/\|w\|$. Maximizing it means minimizing $\|w\|$ and due to the monotony of the norm $\|w\|^2 = w^T w$.

Of course, it can happen that no hyper-plane is able to classify all points correctly. Slack variables ξ_k are therefore introduced. The trade-off between the best possible classification of the data-points (the contribution of the slack variables) and the maximizing of the margin is controlled by the box contraint parameter C.

New data-points (queries) are estimated as

$$\texttt{SVM}(x) = w \cdot x + b. \tag{2.12}$$

The classification is given by the side of the hyper-plane on which the data-point resides which corresponds to taking the sign of this estimation.

In comparison to other machine learning algorithms like neural networks for example, the SVMs present the big advantage of taking the form of a convex optimization problem. But their greatest strength in my opinion is the ability to use transformations for representing a non-linear separation. Therefore, we have to to work in the dual space, where the SVM optimization problem now reformulates

$$\begin{aligned} \underset{\alpha_i}{\text{maximize}} \quad & \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C \; i = 1, \dots, N \\ & \sum_{i=1}^{N} \alpha_i y_i = 0, \end{aligned} \tag{2.13}$$

and the estimation of a new data-point now becomes

$$\texttt{SVM}(x) = \sum_{i=1}^{N} \alpha_i y_i (x_i \cdot x) + b. \tag{2.14}$$

The dual formulation gives its name to support vector machines. A new data-point is estimated based on some vectors of the training size and their corresponding weights. The box constraint parameter here corresponds to the maximum weight of a support vector. In the case of privacy-friendliness, support vectors are very sensitive data as they directly represent instances from the original training data-set.

2.3.2 Mercer's trick and non-linear support vector machines

The main problem of support vector machines as described above is that they are limited to linear relations between the feature elements. However, feature relations, especially in high dimension spaces, require non-linear relations. Therefore, it would be interesting to map the instances into a space of higher dimension $\phi(x)$.

The feature vectors are only appearing in the form of a scalar product, which allows us to use Mercer's trick. The idea is to replace the scalar products by a kernel function $K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$. In most kernel functions, the feature vector is mapped into a space of infinite dimension. We thereby have to consider the scalar product in a Hilbert space to keep the mathematical sense of it: $\phi(x_i) \cdot \phi(x_j) = \langle \phi(x_i), \phi(x_j) \rangle_{\mathcal{H}}$.

We can define the kernel matrix as $K_{ij} = K(x_i, x_j)$. A very interesting property of the kernel matrix is there is no need for computing the explicit transformation of each data-point as we are only interested in their scalar product. Indeed, they are never used on their own, but always in the kernel matrix. As such, one may directly compute the direct matrix. This is even more interesting as the scalar product in Hilbert spaces are practically unfeasible as such. Different kernel functions are possible, that represent the scalar product of different transformations. These kernel functions rely more often on one or more hyper-parameter that has to be trained. The most common choices are

- **Linear kernel function**: this corresponds to computing directly the scalar product onto the identity transformation $\phi(x_i) = x_i$ and is thus equivalent to a linear support vector machine.

$$K(x_i, x_j) = x_i \cdot x_j. \tag{2.15}$$

- **Radial basis kernel function (RBF)**: this kernel function has one hyper-parameter. At a scale factor excepted, this corresponds to the probability of the second instance given the first one with standard deviation σ. An interesting property of this kernel function is its ability to measure similarity as it converges to zero as the distance between both data-points increase and the point thus becoming more and more dissimilar. RBF are known to have excellent generalization capabilities.

$$K(x_i, x_j) = e^{\frac{-\|x_i - x_j\|^2}{2\sigma^2}}. \tag{2.16}$$

- **Polynomial kernel function**: this kernel function also has one hyper-parameter d. In practice, $d = 2$ is often chosen as higher order tend to overfit.

$$K(x_i, x_j) = (x_i \cdot x_j + 1)^d. \tag{2.17}$$

Due to their strong generalization power, RBF kernel generally perform better than other kernels, especially if no additional knowledge of the data is available or multi-class problems as the different classes often are often more diverse and require more generalization. In the case of intrusion detection systems, RBF kernel functions also tend to show better results [47]. Though, other kernels seem to show better results in some specific binary cases [36]. In our case and as we will work with binary classifiers, but multi-class classifiers, we will also opt for RBF kernel functions.

The SVM dual formulation now becomes

$$\begin{aligned} \underset{\alpha_i}{\text{maximize}} \quad & \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C \; i = 1, \ldots, N \\ & \sum_{i=1}^{N} \alpha_i y_i = 0, \end{aligned} \tag{2.18}$$

with estimation given by

$$\texttt{SVM}(x) = \sum_{i=1}^{N} \alpha_i y_i K(x_i, x) + b. \tag{2.19}$$

However, there is also a downside: we are now facing an optimization problem in the dimension of the number of input instances and not of the dimension of the feature space anymore. This means that the number of support vector will drastically increase, which is a bad scenario for our privacy-friendly models. This is one of the investigated trade-offs.

2.3.3 Multi-class SVMs

The main problem of SVM for multi-class classification is that they are not natively suited for it. Indeed, SVM return a single number representing how far the tested data-point is from the hyper-plane. SVMs are by definition created for binary classification as the whole idea behind it is to separate the feature space into two parts. The kernel trick doesn't change anything to that as it only projects the data-point into a space of higher — often unlimited — dimension where a new hyper-plane is searched for, but still dividing this new higher dimension space into two parts.

Ideas for classifying SVMs into different classes could be for example to put the output number into bins. However, this is a very bad idea as it would in fact suggest that all the different classes are divided by parallel hyper-planes at a distances corresponding the the range of the bins, which range could also be trained (figure 2.4a). It almost never occurs that one hyper-plane separates two classes perfectly, the fact that $n - 1$ parallel hyper-planes separate the feature space according the the n different classes is even less likely. This hypothesis is much too strong and must be discarded. We therefore have to use more than one SVM and combine them.

We will present here two different ways of combining SVMs that have both been tested in the case of intrusion detection systems [47], [83] and produce fairly similar results although they never have been compared using exactly the same model for the rest.

The first way and the most frequent one is to create n different classifiers, each for a class, and to look at which one produces the best results:

$$\min \{\mathtt{SVM}_k(x_i)\}_{1\dots n}, \tag{2.20}$$

where $\mathtt{SVM}_k$ represents the SVM for each class k. Each classifier takes as binary input, +1 for the class and -1 for all the rest data-points (figure 2.2). A way of interpreting it is looking for which SVM the data-point lies the farthest from the hyper-plane, at its good size (i.e. positive). This can be graphically observed at figure 2.4b. This is called the *one-against-all* model.

Another way of doing it is to work with successive SVMs as in figure 2.3 and is referred to as *tree-based* models in this thesis. The first SVM is trained using one class as input versus all the others, as the first model. However the next classes are trained using one of the remaining classes versus the others remaining classes. The big advantage is that is only need $n - 1$ different SVMs, which is one less compared to the previous model. Another advantage is that the last SVMs are trained one more specific data. However, the drawback is – which is recurrent in series model — that if one of the SVM isn't efficient, all the model is suffering. Graphically, this can be interpreted as first dividing the space with an hyper plane into two parts. One of the two parts is the subdivided into two new parts and so on (figure 2.4c). Other tree structures are also to be considered, but this is not the scope of this thesis. We will limit ourselves to the consideration of these tree-structures in different orders. The goal of the machine-learning study of this thesis is to try to reduce the number of operations made by the evaluation of a query on the machine-learning algorithm. Computing 4 SVMs instead of 5 indeed decreases these number of computations. We thus want to

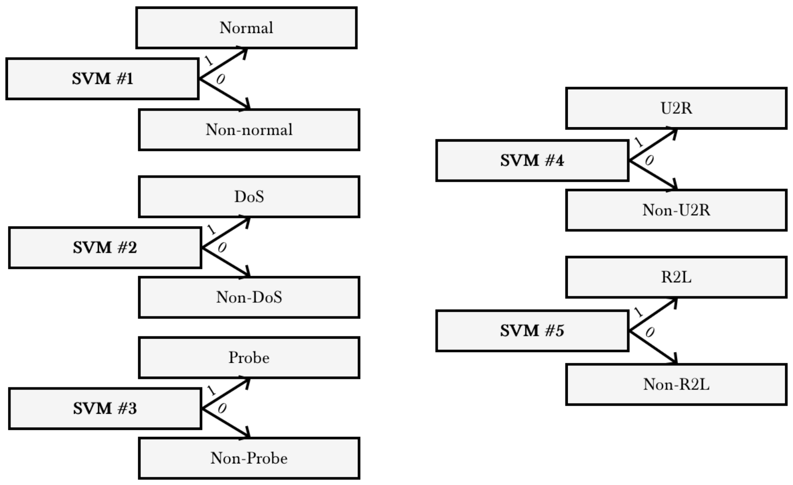


FIGURE 2.2: One-against-all multi-class SVM model.

investigate if this has an impact on the classification performance. A comparison study between the different tree-based models is a totally different topic which is not pursued here. We just limit ourselves to the different models found in the papers.

For the sake of completion, we also have to mention the existence of *one-against-one* models where a binary classifier is trained for all combinations of two classes. The winning class it the one whose corresponding classifiers have the most positive outputs. However, this model needs $n_c(n_c - 1)/2$ different classifiers where n_c is the total number of classes, which is the opposite of the goal we are seeking.

2.4 Nearest neighbours

A classification algorithm that is often used in intrusion detection systems is the k-nearest neighbour (k-NN). This is a simple and effective method based on the distance of the elements in the feature space. Intuitively, when we want to characterize new data elements, we compare it to existent data elements and try to assign to it the characteristics — in this case, the intrusion target — of the similar ones. In other words, the test set elements are classified as similar training elements. Though, how do we count elements to be similar, how do we measure similarity? In the k-NN algorithm, the elements are considered as data-points in the feature space and the similarity is measured as the euclidean distance between them. The most common target of the similar elements is then assigned to the test element.

More concretely, let's first consider a binary classifier of observations and targets $(x_1, y_1), \ldots, (x_n, y_n)$ where observations $x_i \in \mathbb{R}^d$ and targets $y_i \in \{0, 1\}$. To classify a new observation (x_i, y_i), we just have to compare it to the k other nearest observations and assign to the test observation the dominant target among its k neighbours. In the classical k-NN algorithm, the neighbours are defined by the Euclidean distance.

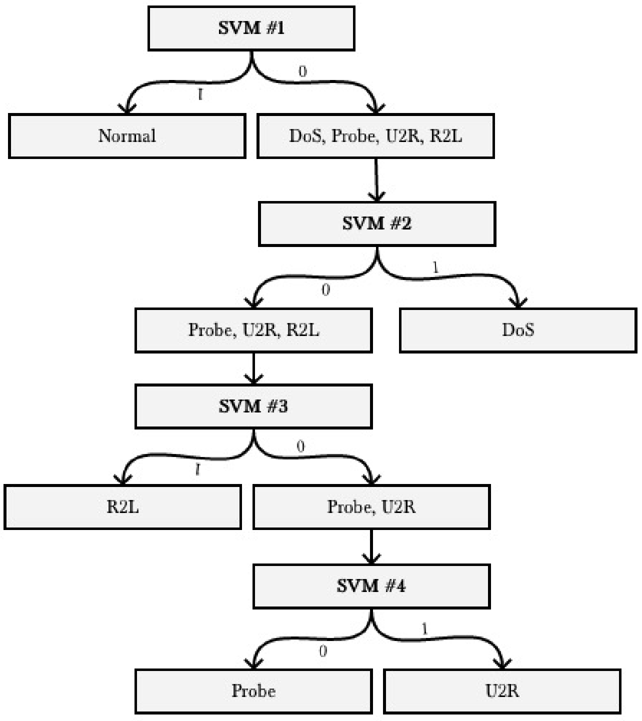
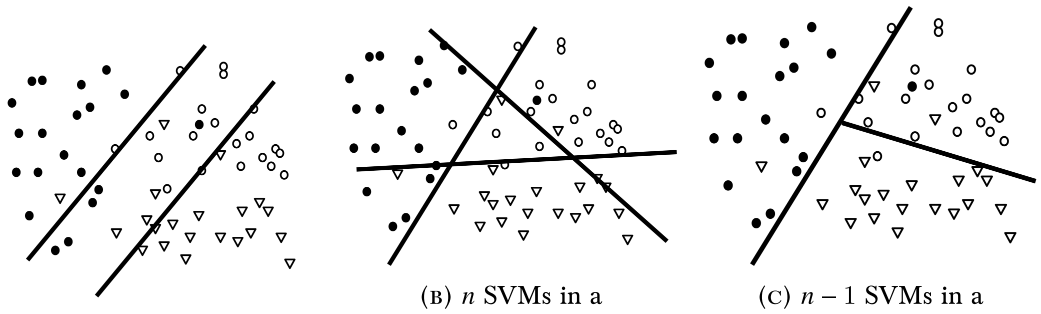


FIGURE 2.3: Tree-based multi-class SVM model.



(A) Single SVM with bins. (B) n SVMs in a one-against-all model. (C) $n-1$ SVMs in a tree-based model.

FIGURE 2.4: Comparison of different multi-class models in the feature space. In this case, there are three classes $n = 3$: dots, circles and triangles.

However, because of the monotony of the square function, one may use the square of the distances instead, avoiding to compute an extra square root, which is very useful in frameworks where operations are expensive like MPCs:

$$\mathrm{d}^2\left(x_i, x_j\right) = \left\|x_i - x_j\right\|^2 = \sum_{k=1}^{d}\left(x_{ik}, x_{jk}\right)^2 . \tag{2.21}$$

Unlike binary classification algorithms like SVMs, k-NN algorithms are suited for multi-class problems without requiring any construction. However, they have also been used for binary classification in a construction [1].

2.4.1 Condensed Nearest Neighbours Data Reduction

The problem of the nearest neighbours algorithm is the time it takes to compute for large data-bases. First, the computation of the distances increases linearly with the number of training points for each test points. Secondly, finding the nearest neighbours requires at least each distance to be touched once, a linear complexity being thus here a lower bound. As we will see, one of the main trade-offs of MPC is the drastic increase in execution time. Therefore, the reduction of data-points becomes a necessity for the use of MPC-based k-NN.

The *condensed nearest neighbours data reduction* (CNN) algorithm aims at finding the only relevant data-points in the whole training set. The unclassified data-points then only need to be compared with the prototype points and not the whole data-base anymore. It is based on the idea that misclassified data-points lie close to the decision boundary, and one can just keep the data-points near the decision boundary and discard the others. The goal of the algorithm is to classify all training data-points into three categories:

- **Outliers**: points which would not be recognized as the correct type if added to the data-base later. In other words, these data-points are the ones making the model perform worse with them than without. They increase the complexity of the decision boundary and increase significantly the number of kept data-points and thus the relevance of the algorithm as it tries to keep a minimum of points defining the decision boundary. These points are to be discarded.

- **Prototypes**: the minimum set of points required in the training set for all the other non-outlier points to be correctly recognized. These data-points thus contain almost all of the relevant data. These points define the decision boundary and are to be kept.

- **Absorbed points**: points which are not outliers, and would be correctly recognized based on the sole prototype points. This could also be characterized as the redundant data. These points are far from the decision boundary and are to be discarded.

The outliers are first found by testing all existing points against the rest of the data-base with the chosen number k of neighbours. If the point isn't recognized as the correct class, it is considered as an outlier, otherwise not. After defining all the outliers, one can proceed into the classification the remaining data-points as prototypes or absorbed points. This is done as following. Each point is once again tested against the data-set without the outliers, but with one sole neighbour, i.e. $k = 1$. If it is well classified, it is considered as an absorbed point and can be left out of the final data-set. Otherwise, it is considered as a prototype ans is considered relevant to the classification or not redundant. These data-points are the sole ones making it to the final data-set. All points are tested until no prototype anymore makes it into the final data-set. The CNN data-reduction is given in algorithm 2.

In the condensed nearest neighbours algorithm, all new data-points are tested against the reduced data-base. Another big advantage of this algorithm is that the algorithm now always has to be used with $k = 1$ because of the way we defined our prototypes and the final algorithm is thus faster. However, the classification with CNN will most of the time lead to a slightly different classification than with the classical k-NN against the whole data-set.

```
Data: Set $\mathcal{S}$ of $n$ feature points $\{x_i\}_{i=1\ldots n}$ and corresponding targets $\{y_i\}_{i=1\ldots n}$,
      number of neighbours $k$ for outlier detection
Result: Reduced set $\mathcal{R}$ of $m < n$ feature points $\{x_i\}_{i=1\ldots m}$ and corresponding
        targets $\{y_i\}_{i=1\ldots m}$
foreach $(x_i, y_i) \in \mathcal{S}$ do
    $t \leftarrow$kNN($k$, $x_i$, $\mathcal{S}$)
    if $t \neq y_i$ then remove chosen $(x_i, y_i)$ from $\mathcal{S}$
end
$\mathcal{R}_0 \leftarrow \emptyset$
add random $(x_i, y_i)$ to $\mathcal{S}$
repeat
    $\mathcal{R}_1 \leftarrow \mathcal{R}_0$
    for $(x_i, y_i) \in \mathcal{S}$ do
        $s \leftarrow$kNN($k = 1$, $x_i$, $\mathcal{R}_1$)
        if $s \neq y_i$ then add $(x_i, y_i)$ to $\mathcal{R}_1$
    end
until $\mathcal{R}_0 = \mathcal{R}_1$
return $\mathcal{R}_1$
```

Algorithm 2: The condensed nearest neighbours algorithm. This algorithm relies on an implementation of the k-NN, represented here by the `kNN` function that takes as input the number of neighbours k, the data-points to be classified x_i and the set in which it should search for the neighbours $\mathcal{S}$.

Chapter

3 Achieving the preservation of privacy

3.1 Different approaches on privacy-friendliness

In a world of constant data exchanges between different entities, it might be useful to develop methods not only to protect data during the transit as encryption classically aims to, but also when in possession of an entity that should not be able to read all of that data. This is of particular interest for computation outsourcing where a specific data-set has to be processed by an external entity that should not be able to infer anything more than what is asked from it. Furthermore, it might even be wished that one or more external entities might not be able to read the output of their computations; it should solely be readable by the original owner(s) of the data. This is of particular interest for the — now everywhere — cloud solutions. A protocol that respects the private character of data when treated by other entities than the owner is called *privacy-friendly* or *privacy-preserving*. There are different approaches to privacy-friendliness and for the sake of completeness, hereafter follows a short survey, which also justifies our choice.

3.1.1 Differential privacy

Instead of encrypting all the data, one could alternatively directly address the core problem of why we want the data encrypted: to prevent other parties to get any sensitive information contained in the data. In the case that will interest us in this thesis, we will be in possession of a lot of data from personal users which is confidential and can therefore not be traced back to the user. In 2009, Netflix launched the Netflix prize on data recommendation: the first group to improve the recommendation score of Netflix by 10% or more would win 1.000.000$. They provided a data-set to let the participants train their models. They also took care of anonymizing the data before they made it accessible. However, Narayanan and Shmatikov showed how they could re-identify a lot of the users by using the scores of the users on IMDb [57]. *Differential privacy* addresses this problem by adding noise to the data and thereby achieving a better anonymization [34]. In this way, the data is still usable for statistical models but cannot be used to identify anyone as easily as before. Still, differential privacy is limited by the fundamental and intrinsic relation between anonymization and statistical relevance.

One cannot obtain the first without inevitably having an influence on the second one, and reciprocally.

3.1.2 Homomorphic encryption

Differential analysis is a statistical approach of anonymity, but there also exist some cryptographic approaches, where the external entity is not able to read the results of what it produces. It is possible to construct a protocol with one or more third parties in a way that they cannot possibly learn anything from what they are receiving nor what they are sending back: the information is processed in an encrypted and not a clear form. Encryption schemes that allow mathematical operations to be executed on the encrypted data are called *homomorphic* and were first proposed by Rivest et al. in 1978 [68]. For example, the RSA encryption scheme preserves the multiplication over the encrypted data [69]. The RSA encryption scheme is given by $\mathscr{E}(m) = m^e \mod N$. We thus have $\mathscr{E}(m_1) \cdot \mathscr{E}(m_2) = \left(m_1^e \mod N\right)\left(m_2^e \mod N\right) = (m_1 m_2)^e \mod N = \mathscr{E}(m_1 m_2)$. Unfortunately, this property is only true for multiplication and is therefore quite limited in its applications. We therefore refer to RSA as a *somewhat-homomorphic* encryption scheme (SHE). When all mathematical operations are possible, we say from the encryption scheme that it is *fully-homomorphic* (FHE). Up to now, only some schemes based on finite fields possess this property.

The most accomplished method up to now is called the *Approximate Eigenvector Method* and is based on the *Learning With Errors* (LWE) encryption scheme. This scheme is also known for still being secure in a post-quantum era. If C_1 and C_2 are two matrices with common eigenvector s, we notice that the sum or multiplication of their respective eigenvalues m_1 and m_2 corresponds to the eigenvalue of the sum or multiplication of C_1 and C_2 with respect to s. The eigenvector act as a private key and the eigenvalues as the secret messages. The scheme is thus fully homomorphic. However, eigenvectors are easy to find and the scheme is thus also insecure. To resolve this, the method uses approximate eigenvectors $sC = ms + e \approx ms$ which is known to be still solvable in finite fields under a few assumptions about the error e [66].

3.1.3 Multi-party computation

When different parties participate to the input, the homomorphic encryption described above cannot be used anymore: all parties have to share the same secret key which keeps their data private with respect to the third party, but not to each other. Multi-party computation addresses this problem. Furthermore, it also allows the parties to compute a common function on their private inputs without needing one or more third parties.

More concretely, let us now imagine a problem where the goal is to compute some common function f over private inputs x_i. In other words, we want to compute $f(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$ where each input x_i is privately provided by party P_i, which ultimately learns y_i and nothing more: not the other outputs, nor the other inputs. A first naive implementation would be to trust a third party to privately receive each player's input, computing the function and privately communicating the corresponding responses to the concerned parties. However, it also possible to obtain the same results without the trust of a third party, where only the parties are participating to the protocol. This is called *multi-party computation* (MPC) also referred to as *secure multi-party computation* (SMC). This approach has been chosen to solve our problem and will be described more extensively here under and in the next section.

Multi-party computation originated with the toy example presented by Yao in 1982 [94] that is now known as the *Millionaire's Problem*: two millionaires both want to know who is richer, but none of them wants to disclose their fortune to the other one nor trust a third party. Other applications of MPC may concern electronic voting or solutions of private-data as a service (PDaaS). The different methods defined here under have a common general way of working. The computation is done in rounds: each party computes some algorithms on its own and then exchange som information with the other parties. The exchanges do not happen continuously, but all together in what is called a round. A new computation phase can then take place followed by a next exchange phase.

Bit-wise decomposition

There basically exist two big families of multi-party computation protocols and each of them is based on a different cryptographic primitive. A cryptographic primitive is the basic block or idea on which the whole protocol is based. The first one described here decomposes every operation into a boolean circuit and every value into its bits, hence its name. The cryptography takes place at bit-level. The two multi-party protocols described here under use the idea of oblivious transfer, which is first described.

Oblivious transfer The idea behind *oblivious transfer* originally described by Rabin in 1981 [65] is the transfer of an certain information in possession of a first party and asked by a second party, without the first party knowing which information has been transferred. Hence the name oblivious, or alternatively, unconscious. Different protocols exist and are all based on the *RSA scheme*.

The most common version is the *1-2 oblivious transfer* and goes as follows [37]: Alice is in possession of two messages m_0 and m_1 and Bob wants to get message m_p. Alice first generates a set of private key d and public key (N, e) and sends two random messages x_0 and x_1 to Bob. He then generates a random message k and encrypts it with the x corresponding to the wanted message: $v = (x_p + k^e) \mod N$ and sends it to Alice. She then recovers both k without knowing which one corresponds to Bob's original one: $k_i = (v - x_i^d) \mod N$. These k_i then serve to encrypt the messages which are finally sent to Bob $s_i = m_i + k_i$. Bob can then only decrypt the wanted message $m_p = s_p - k$.

This protocol has been generalized to more than two parties [44], [78], [84].

Yao's garbled circuits *Garbled circuits* (GC) were first introduced by Yao in 1986 and are now one of the most efficient solutions for generic secure two-party computation [95]. A function has to be decomposed into a boolean circuit consisting of two-input gates (e.g. XOR and AND). Let's consider the simplest example of evaluating an AND-gate between Alice and Bob. Alice first generates a different random sequence — also called *labels* — for each possible value of each input — also called *wires* — and output. In the truth table, the output are then symmetrically encrypted with the hash of each

A	B	output
0	0	0
1	0	0
0	1	0
1	1	1

(A) Truth table.

A	B	output
x_A^0	x_B^0	x^0_{output}
x_A^1	x_B^0	x^0_{output}
x_A^0	x_B^1	x^0_{output}
x_A^1	x_B^1	x^1_{output}

(B) Labelled truth table.

output
$\mathscr{E}_{H(x_A^1\|x_B^0)}\left(x^0_{\text{output}}\right)$
$\mathscr{E}_{H(x_A^1\|x_B^1)}\left(x^1_{\text{output}}\right)$
$\mathscr{E}_{H(x_A^0\|x_B^0)}\left(x^0_{\text{output}}\right)$
$\mathscr{E}_{H(x_A^0\|x_B^1)}\left(x^0_{\text{output}}\right)$

(C) Garbled output.

TABLE 3.1: Garbling of the AND-gate.

corresponding input. These four resulting cyphertexts are then randomly permuted — hence the name *garbled* — and sent to Bob. The garbling of an AND-gate is illustrated at table 3.1. Once Bob receives the garbled gate, he then asks Alice for her label. As they have been randomly chosen, she can send it to him without him possibly knowing what value it corresponds to. Afterwards, he also needs to know the label of his input. This part is a bit more tricky and is solved using the previously described ***oblivious transfer***. Bob can now compute the hash of the two labels and decrypt each element of the garbled gate until he finds one corresponding with the garbled gate he received from Alice. He can then reveal the value to Alice or she can reveal the mapping of the garbling. The same principle can be used on a multi-gate circuit by garbling the sole end result.

It is interesting to note that the secure evaluation of a sole AND-gate does not respect the principles of multi-party computation, by definition of the AND-gate. Indeed, if the final solution is 1, both players know their respective input value, which is thus disclosed[1]. Therefore, the total functions evaluated cannot be injective for any input. The circuit corresponding to the Millionaire's problem is given at figure 3.1 and while it consists of AND-gates, the function isn't injective at all with respect to each millionaire's fortune.

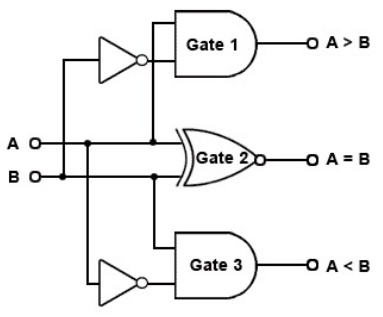


FIGURE 3.1: The digital comparator is the boolean circuit used to solve Yao's millionaire problem.

This protocol executes in polynomial time, but there exist a lots of optimiza-

[1]This is not the case for the XOR-gate as the 1-output has two corresponding inputs possible, as has the 0-output

tions that allow to garble and evaluate the gates more rapidly [92].

GMW PROTOCOL The *Goldreich-Micali-Widgerson* (GMW) protocol can be seen as an extension of garbled circuits to multiple parties using Yao's idea of using oblivious transfer [39]. The main principles here are based upon bit-sharing: each party shares its input bits among the n players $b = \sum_{i=1\dots n} b_i \mod 2$. Each player then processes their shares among the circuit. XOR-gates are easy as they can just addition the shares $c_i = a_i + b_i$. However AND-gates are more tricky: we can see from the decomposition that $c = a \cdot b = \sum_{i \neq j} a_i b_j + \sum_{1 \leq i < j \neq n} \left(a_i b_j + a_j b_i\right) \mod 2$. Each party will then have to compute $a_i b_i + \sum_{i \neq j} \left(a_i b_j + a_j b_i\right)$. As for the XOR-gate, the first part is trivial to evaluate, however, the second cannot be computed by party i without more information from party j. This is solved by using a variant of garbled circuits with oblivious transfer between parties i and j.

Avoiding bit-wise decomposition

Alternatively to boolean circuits, arithmetic circuits can also be used for multi-party computation. The problem of bit-wise decomposition and the use of the boolean circuit transcription of the function that we jointly want to evaluate, is their expensiveness. Indeed, a simple operation can rapidly lead to a lot of gates. For example, let's consider the addition of two numbers: for two numbers of n bits, the total number of gates is $5n$ in a full adder composition. This is even worse for multiplication. Of course, some optimizations can be made, but the general number of gates is very high compared to the arithmetic circuit of the same function, where it would just be one single gate for addition and multiplication. In general, MPC over arithmetic circuits require a much higher number of rounds. Though, they seem in total more efficient for solving complex problems. Even 2-party comparisons do not seem much more efficient with garbled circuits — for which they were originally tailored – compared to the method described here-under [14].

The cryptographic primitive of multi-party computation over arithmetic circuits is based upon *secret sharing*. Each party computes its own version of the circuit with the shares of the different parties. At the end of the circuit processing, each party has a share of the final output, which can then be put together to obtain the final output. We first have to define a way for a party to share its secret among n parties.

ADDITIVE SECRET SHARING The simplest idea is just to divide the secret a in n shares a_i using a simple summation: $a = \sum_{i=1\dots n} a_i$. This cryptographic primitive is often referred to as the *linear secret sharing scheme* (LSSS). However, doing it in this manner allows the shares to release some information about the secret, as the shares are not random and strongly depend on the secret. They are two solutions to this problem and the first one is to consider additive sharing over $\mathbb{Z}_q$. The sharing now becomes $a = \sum_{i=1\dots n} a_i \mod q$ which solves the problem, as the shares can now really be chosen at random. The other solution is over $\mathbb{Z}$ and consists of choosing a sufficiently large interval in which the shares are chosen to dilute sufficiently the statistical information about the secret, typically $a_i \in [-A2^\kappa, A2^\kappa]$ with A the size of the interval of the secret

$a \in [-A, A]$. The size of a is of maximum bit-length k and we therefore also refer to $\mathbb{Z}_k$ alternatively the interval.

Another consequence of this scheme is that all parties are vital to the recovery of secret, as the loss of one secret prevents us to reconstruct the secret or any statistical information about it. The scheme does not tolerate the loss or treason of one party and is therefore very sensitive to any failure or malicious player. This problem is solved by polynomial secret sharing.

Polynomial secret sharing The idea of polynomial sharing was originally proposed by Shamir in 1979 [76]. This cryptographic primitive is often referred to as *Shamir secret sharing*. It allows n parties to share a secret in such a way that any subset of $t + 1$ parties can later reconstruct the secret, but any subgroup of maximum t parties cannot do so. The scheme is based on the single fact that for any polynomial of degree d, any subset of $d + 1$ or more different points can be used to reconstruct the polynomial completely whereas any subset of at most d points lefts us with an infinite number of possibilities.

The scheme goes as follows. The party that wants to share its secret first constructs a polynomial of degree t:

$$h(z) = a + \sum_{i=1}^{t} b_i z^i, \tag{3.1}$$

with secret a random coefficients b_i. For the same reasons as for additive secret sharing, the coefficients can either be chosen in $b_i \in \mathbb{Z}_q$ (which also leads to the consideration of polynomial $h(z) \mod q$ instead), either in the interval $b_i \in [-A2^\kappa, A2^\kappa]$.

We verify that $h(0) = a$ and distribute shares a_i to each party i as follows $a_i = h(i)$. $t + 1$ parties can now reconstruct the polynomial together using e.g. Lagrange's polynomials and compute $f(0)$ to recover the secret share. An interesting property is that the recovery can be done as a simple linear combination. Indeed, we have

$$\begin{aligned} h(0) &= \sum_{i=1}^{n} l_i(0) h(i) && (3.2) \\ a &= \sum_{i=1}^{n} r_i a_i, && (3.3) \end{aligned}$$

where $r = (r_1, \dots, r_n) = (l_1(0), \dots, l_n(0))$ is called the ***recombination vector*** with $l_i(z)$ the i-th order Lagrange polynomial, for example

$$l_i(z) = \prod_{1 \leq j \leq n, j \neq i} \frac{z - j}{i - j}.$$

The sum of a the elements of a recombination vector always equals 1 ($\sum_{i=1}^{n} r_i = 1$). It can be constructed with any generating set of the polynomial space $\mathbb{Z}[z]$ (or alternatively $\mathbb{Z}_q[z]$), as long all parties use the same set.

3.2 Operations with additive secret sharing

From now on, we will use the bracket-notation for a secret $\langle \bar{a} \rangle$ in the space of secrets $\mathbb{Z}_{\langle k \rangle}$. The shares are denoted by $\langle \bar{a}_i \rangle$ an the bar on top is there to indicate that the secret represents an integer. No brackets means that the value is not secret and thus known to everybody.

3.2.1 Performing basic operations

We will now look at how to perform the basic operations that are at the core of every arithmetic circuit in polynomial secret sharing, i.e. addition and multiplication, with a public value or between secrets.

Addition and multiplication of public constants

Let us first consider the case where a shared value is multiplied or added to constant, i.e. $\langle \bar{b} \rangle = \langle \bar{a} \rangle + c$ and $\langle \bar{b} \rangle = c \cdot \langle \bar{a} \rangle$. This case is simple: each party just adds or multiplies his share with the constant. This rests on some arithmetic properties of polynomials: one can easily verify that $\langle \bar{a}_i \rangle + c$ is a share of $\langle \bar{a} \rangle + c$ and $c \cdot \langle \bar{a}_i \rangle$ of $c \cdot \langle \bar{a}_i \rangle$.

Addition of two secrets

Let us now consider the addition of two hidden values $\langle \bar{d} \rangle = \langle \bar{a} \rangle + \langle \bar{b} \rangle$. This can be done by just adding the respective shares of secrets $\langle \bar{a} \rangle$ and $\langle \bar{b} \rangle$: $\langle \bar{d}_i \rangle = \langle \bar{a}_i \rangle + \langle \bar{b}_i \rangle$. This is a direct consequence of equation 3.3.

Multiplication of two secrets

Unfortunately, it is not possible to adopt the same strategy for multiplication as the multiplication of two polynomials of order t will lead to a new polynomial of order $2t$ which will double the number of parties needed to recover the secret output. In real-case functions with a lot of multiplications, this becomes rapidly impracticable and theoretically unsolvable if the degree of the new polynomial exceeds n. Another problem is of a statistical order: the new polynomial is not random anymore as it is for example not irreducible anymore by construction.

A solution to this problem is to use multiplication triples. Each triple is a set of three pre-computed values $\{\langle \bar{x} \rangle, \langle \bar{y} \rangle, \langle \bar{z} \rangle\}$ that verify $\langle \bar{z} \rangle = \langle \bar{x} \rangle \cdot \langle \bar{y} \rangle$. These multiplication triples are generated before the algorithm starts to evaluate the circuit. This is called the *offline phase*. The part where the algorithm evaluates the arithmetic circuit in itself is called the *online phase*. The computation of these triples requires fully homomorphic encryption and will not be further considered in this thesis (the exact method used in this thesis is based on *Beaver triples* [9], [62]). Using these triples, we can now easily compute a multiplication of two arbitrary secret values $\langle \bar{a} \rangle$ and $\langle \bar{b} \rangle$. The protocol is given in algorithm 3. The computation of secret $\langle \bar{e} \rangle$ and $\langle \bar{d} \rangle$ corresponds to the security of a one-time pad which has an perfect theoretical security. In other words, if the pad — multiplication triple here — is statistically secure, $\langle \bar{e} \rangle$ and $\langle \bar{d} \rangle$ are also secure and can be revealed without revealing anything on the input or the triple. Finally, we can algebraically verify that $\langle \bar{c} \rangle$ results in the correct output. The trick here is to use the one-time pad to allow to reveal e and d. This allows to avoid making a multiplication of two shared values which in turn avoids augmenting the degree of the underlying polynomial used for the sharing. Multiplication can thus be performed using polynomial secret sharing, but at the cost of having to use fully homomorphic encryption for the generation of multiplication triples and one extra round to reveal the values of $\langle \bar{d} \rangle$ and $\langle \bar{e} \rangle$.

In practice, specific *square pairs* are also used to perform squaring operations in a similar manner, instead of using the multiplication described here.

```
**Data**: Multiplication tripe $\{\langle \bar{x} \rangle, \langle \bar{y} \rangle, \langle \bar{z} \rangle\}$ and inputs $\langle \bar{a} \rangle$ and $\langle \bar{b} \rangle$
**Result**: Product $\langle \bar{c} \rangle = \langle \bar{a} \rangle \cdot \langle \bar{b} \rangle$
$\langle \bar{e} \rangle \longleftarrow \langle \bar{x} \rangle + \langle \bar{a} \rangle$
$\langle \bar{d} \rangle \longleftarrow \langle \bar{y} \rangle + \langle \bar{b} \rangle$
$e \longleftarrow$ Open($\langle \bar{e} \rangle$)
$d \longleftarrow$ Open($\langle \bar{d} \rangle$)
$\langle \bar{c} \rangle \longleftarrow \langle \bar{z} \rangle + e\langle \bar{a} \rangle + d\langle \bar{b} \rangle - ed$
**return** $\langle \bar{c} \rangle$
```

Algorithm 3: Secure multiplication protocol. The function Open refers to the revelation of its input. In the active-case model, this also comprises a verification step (cf. infra).

3.2.2 Performing more advanced operations

The last operation we should be able to perform to run most of the algorithms with additive secret sharing is the comparison and the conditional assignment. Conditional assignment is the assignation to a secret value of one of two secret values depending on a secret condition, without knowing which of the two has been assigned. However, the comparison of two integers is not straightforward and is based on some other protocols, which we first need to describe. The first thing to note is that additive secret sharing has no support for secret boolean values, and uses a secret integer value of 1 and 0.

Modulo

The modulo protocol computes the value of a secret value $\langle \bar{a} \rangle \in \mathbb{Z}_{\langle k \rangle}$ modulo 2^m or in other words, its first digits in base 2. More specifically `Mod2m`$(\langle \bar{a} \rangle, k, m) = \langle \bar{a} \rangle / 2^m$. The protocol is given at algorithm 4 and rests on two sub-protocols `BitLT` and `PRandM` [4], [20].

The `PRandM` protocol generates two random values $\langle \bar{r'} \rangle$ (of length $l_1 = k + \kappa - m$) and $\langle \bar{r''} \rangle$ (of length $l_2 = m$) where the second is also given in its bit-wise decomposition $\langle \bar{r''} \rangle_B$. Similarly as the multiplication, the generation of these resides in the pre-sharing of a bit list in the offline phase. To construct the bit-wise decomposition of these values, each player just has to pick the right amount of bits l in the shared list. We can then reconstruct the non-decomposed share with

$$\langle \bar{r} \rangle \leftarrow \sum_{i}^{l} 2^i \cdot \langle \bar{r_i} \rangle_B. \tag{3.4}$$

For the first value $\langle \bar{r'} \rangle$ of length $l_1 = k + \kappa - m$, the bits can be discarded after this computation and while they are kept for the other share $\langle \bar{r''} \rangle$ of shorter length $l_2 = m$.

The reason we have to use these shared random values is because the whole operation on which the modulo resides here is BitLT and only works with bit-wise decompositions. Though, the value $\langle \bar{a} \rangle$ we want to compute the modulo of is not given in bit-wise decomposition. Similarly as we did for addition, we use a intermediate value c that we can reveal. This allows to easily compute the modulo and then the bit-wise decomposition.

In the begin of this section, we said that arithmetic circuits avoided the bit-wise decomposition unlike e.g. garbled circuits. However, it seems here that we are still residing on bit-wise decomposition for the comparison. In fact, this protocol does not really use bit-wise decomposition, at least not in the same sense as the schemes based on boolean circuits. A first difference is that the secret values we want to compute operations on, here $\langle \bar{a} \rangle$ are never really decomposed in bits but are only using additions and multiplications. Only c is decomposed into bits during the protocol and only when it is not secret anymore. The other value to be decomposed into bits is $\langle \bar{r'} \rangle$, but this is done by construction and contains no single information on $\langle \bar{a} \rangle$ (in the sense that $\langle \bar{r'} \rangle$ is statistically totally independent from $\langle \bar{a} \rangle$). A second difference here is that the boolean gates are reduced to additions and multiplications. No truth tables are used. We can thus not really talk about boolean operations as they are arithmetic operations in essence. Garbled circuits reside on oblivious transfer and the garbling of encrypted truth tables, which is not the same: not a single arithmetic operations takes place. The sole similarity is that both decompose a comparison into a boolean circuit, but these circuits are computed differently and the bit-wise comparison in itself doesn't happen with the same inputs.

```
Data: $\langle \bar{a} \rangle$, field size $k$ and $m$
Result: $\langle \bar{a'} \rangle = \langle \bar{a} \rangle / 2^m$
$(\langle \bar{r''} \rangle, \langle \bar{r'} \rangle, \langle \bar{r'} \rangle_B) \leftarrow \texttt{PRandM}(k, m)$
$\langle \bar{c} \rangle \leftarrow 2^{k-1} + \langle \bar{a} \rangle + 2^m \langle \bar{r''} \rangle + \langle \bar{r'} \rangle$
$c \leftarrow \texttt{Open}(\langle \bar{c} \rangle)$
$c' \leftarrow c \mod 2^m$
$\langle \bar{u} \rangle \leftarrow \texttt{BitLT}(c'_B, \langle \bar{r'} \rangle_B)$
$\langle \bar{a'} \rangle \leftarrow c' - \langle \bar{r'} \rangle + 2^m \langle \bar{u} \rangle$
return $\langle \bar{a'} \rangle$
```

Algorithm 4: Secure modulo 2^m protocol.

Truncation

Now that we are able to compute the modulo of a secret value, we can easily compute its truncation, which is essentially the opposite of the modulo. Instead of computing the rest of a division by 2^m, we compute the value of this division. This is noted $\texttt{Trunc}(\langle \texttt{a}\rangle, \texttt{k}, \texttt{m}) = \lfloor \langle \bar{a}\rangle / 2^m \rfloor$, where the modulo returns the m last digits $\langle \bar{a}'\rangle$ in base 2, the truncation returns the $k - m$ first digits $\langle \bar{d}\rangle$. This can be easily computed using the rest of a division by 2^m: $\langle \bar{a}\rangle = 2^m \langle \bar{d}\rangle + \langle \bar{a}'\rangle$. The protocol is given at algorithm 5.

```
Data: $\langle \bar{a}\rangle$, field size $k$ and $m$
Result: $\langle \bar{d}\rangle = \lfloor \langle \bar{a}\rangle / 2^m \rfloor$
$\langle \bar{a}'\rangle \leftarrow \texttt{Modm}(\langle \bar{a}\rangle, k, m)$
$t \leftarrow 1/2^m \mod p$
$\langle \bar{d}\rangle \leftarrow t \cdot (\langle \bar{a}\rangle - \langle \bar{a}'\rangle)$
return $\langle \bar{d}\rangle$
```

Algorithm 5: Secure truncation protocol.

Comparisons

The truncation protocol allows now to compute the comparison of two secret values $\langle \bar{a}\rangle, \langle \bar{b}\rangle \in \mathbb{Z}_{\langle k\rangle}$. One just has to observe that the truncation protocols allows to compute the sign of a secret value as the sign is basically just the k-th bit of its bit-wise decomposition. Hence, we can define a lower-than protocol $\texttt{LTZ}(\langle \bar{a}\rangle, k) = -\texttt{Trunc}(\langle \texttt{a}\rangle, \texttt{k}, \texttt{k} - 1)$. The comparison $\langle \bar{a}\rangle < \langle \bar{b}\rangle$ can now be computed as $\texttt{LTZ}(\langle \bar{a}\rangle - \langle \bar{b}\rangle, k)$. The other comparison operators are defined similarly and the equality uses the similar ideas [4], [20].

Conditional assignment

The last protocol we have to design is the secure conditional assignment protocol. It assigns a secret value to a variable depending on the boolean. The protocol described in algorithm 6 assigns secret value $\langle \bar{a}\rangle$ to variable $\langle \bar{c}\rangle$ if $\langle \bar{s}\rangle$ is equal to 1, otherwise it assigning $\langle \bar{b}\rangle$. This protocol will be noted $\langle \bar{c}\rangle \leftarrow_{\langle \bar{s}\rangle} \langle \bar{a}\rangle : \langle \bar{b}\rangle$ further in this thesis [5].

```
Data: $\langle \bar{a}\rangle$, $\langle \bar{b}\rangle$ and boolean $\langle \bar{s}\rangle$
Result: $\langle \bar{c}\rangle \leftarrow \langle \bar{a}\rangle$ if $\langle \bar{s}\rangle = 1$ and $\langle \bar{c}\rangle \leftarrow \langle \bar{b}\rangle$ if $\langle \bar{s}\rangle = 0$
$\langle \bar{d}\rangle \leftarrow \langle \bar{s}\rangle \cdot (\langle \bar{a}\rangle - \langle \bar{b}\rangle)$
$\langle \bar{c}\rangle \leftarrow \langle \bar{b}\rangle + \langle \bar{d}\rangle$
return $\langle \bar{c}\rangle$
```

Algorithm 6: Secure conditional assignment protocol.

3.2.3 Working with decimal numbers

Up to now, we were solely working with secret integers. We can also work with secret decimal numbers.

Representation

In this work, the decimal numbers are numerically represented as fixed points and are noted with a tilde $\langle \tilde{a}\rangle$. A fixed point secret value is represented as a pair of two secret integers: $\langle \bar{a} \rangle$ representing the significand and $\langle f \rangle$ the exponent. The set of secret rational numbers $\mathbb{Q}_{\langle k,f \rangle}$ can defined using the mapping $\langle \tilde{a} \rangle = \langle \bar{a} \rangle \cdot 2^{-f}$ with $\langle \bar{a} \rangle \in \mathbb{Z}_{\langle k \rangle}$. This intuitively corresponds by scaling the range of real values by a value of 2^{-f}. With integers, the security condition was that $k + \kappa < p$, it now becomes $k + \kappa + f < p$.

Basic operations

All operations defined before can now be computed on the significand if the values have the same exponent. When this is not the case, we can easily change the exponent using a scaling protocol as defined in algorithm 7. This can be the case when performing operations between secret fixed point numbers and secret integers — that are in fact secret fixed point values with $f = 0$. It just multiplies the significand if the exponent has to shrink and truncates if it has to grow. When performing the operations on the significands, we just have to prevent that it exceeds the size allowed for the significand and therefore select the right part of it using the modulo for addition (the last digits in base 2) and the truncation for multiplication (the first digits in base 2). The protocols for addition and multiplication are given in respectively algorithm 8 and 9. Comparison is similarly just done on the significands and the same conditional assignment protocol as for secret integers can be used. A more specific protocol can also be used for division as described by Catrina et al. [21].

```
Data: $\langle \tilde{a} \rangle \in \mathbb{Q}_{\langle k,f_1 \rangle}$, $k$, $f_1$ and $f_2$
Result: $\langle \tilde{a'} \rangle \in \mathbb{Q}_{\langle k,f_2 \rangle}$
$m \leftarrow f_2 - f_1$
if $m \geq 0$ then $\langle \tilde{a'} \rangle \leftarrow 2^m \cdot \langle \tilde{a} \rangle$
else $\langle \tilde{a'} \rangle \leftarrow$ Trunc(⟨ã⟩, k, −m)
return $\langle \tilde{a'} \rangle$
```

Algorithm 7: Secure scaling protocol for secret fixed point numbers.

```
Data: significands $\langle \bar{a} \rangle, \langle \bar{b} \rangle$ of $\langle \tilde{a} \rangle, \langle \tilde{b} \rangle \in \mathbb{Q}_{\langle k,f \rangle}$, $k$ and $f$.
Result: significand $\langle \bar{c} \rangle$ of $\langle \tilde{c} \rangle = \langle \tilde{a} \rangle + \langle \tilde{b} \rangle \in \mathbb{Q}_{\langle k,f \rangle}$
$\langle \bar{d} \rangle \leftarrow \langle \bar{a} \rangle + \langle \bar{b} \rangle$
$\langle \bar{c} \rangle \leftarrow$ Mod2m(⟨d⟩, k + 1, k)
return $\langle \bar{c} \rangle$
```

Algorithm 8: Secure addition protocol for secret fixed point numbers.

```
Data: significands $\langle \bar{a} \rangle, \langle \bar{b} \rangle$ of $\langle \tilde{a} \rangle, \langle \tilde{b} \rangle \in \mathbb{Q}_{\langle k,f \rangle}$, $k$ and $f$.
Result: significand $\langle \bar{c} \rangle$ of $\langle \tilde{c} \rangle = \langle \tilde{a} \rangle \cdot \langle \tilde{b} \rangle \in \mathbb{Q}_{\langle k,f \rangle}$
$\langle \bar{d} \rangle \leftarrow \langle \bar{a} \rangle \cdot \langle \bar{b} \rangle$
$\langle \bar{c} \rangle \leftarrow$ Trunc(⟨d⟩, 2k, f)
return $\langle \bar{c} \rangle$
```

Algorithm 9: Secure multiplication protocol for secret fixed point numbers.

3.2.4 Security

The protocol is called secure when adversaries are not able to reveal any information they are not supposed to. In the case of this thesis, this means guaranteeing the privacy of the data. By extension, we then talk about a privacy-friendly algorithm. The additive secret sharing cryptographic primitive is by design secure in the *information-theoretic* model. This means that the information is not protected by the high complexity of an algorithm and thus by the limited computing power of an adversary. Instead, the information is protected by the limited information contained in the data owned by the different parties. Indeed, the cryptographic primitive is based on polynomials in a sense that no group of players of a degree lower than the polynomial can find the secret value. Solving the equation system leads to an infinite number of solutions. This also means that the scheme is cryptanalytically unbreakable: no possible loophole can be found in the scheme.

The only possible attacks are based upon the fact that the shares can release statistical information about the secret. However, this is solved using a much bigger interval than the size of the field in which the secret are defined, using security parameter κ.

Another possible attack resides on the external cryptographic primitives used for the computation of the elements needed for the offline phase, e.g. the multiplication triples. These are based on fully homomorphic encryption that are not information theoretically secure, but *cryptographically secure.* However, this thesis concentrates on the secure multi-party protocols and we will consider them secure.

Passive and active case

A party that uses all its computation power to try to recover the secret, but follows the protocol is called an *honest but curious* adversary. As with all the information that it has, no party can possibly recover the secret, the scheme is secure in the *passive case* also called *semi-honest model.* To protect the system against a *malicious* adversaries, i.e. an adversary that deviates from the protocol intentionally to recover information, shared message authentication codes (MACs) are used to commit each party to its share when it reveals it to everybody. At the public revelation, or opening, the MAC value is reconstructed and everybody checks its correctness. Once the MAC is reconstructed and public, no further operation takes place.

In short, a long-term MAC key α is commonly constructed is such a way that each party receives a share from it $\langle \bar{\alpha}_i \rangle$. This can then be used to commit each player during the addition: in addition to receiving the secret shares $\langle \bar{a}_i \rangle$, each party P_i receives a secret share $\gamma(a)_i$ such that $\alpha \cdot a = \gamma(a) + \ldots + \gamma(a)_n$ (the MAC on a). When performing the addition of their local shares, each party also computes the addition of its shared MACs of the values. Let's consider the addition $\langle \bar{c} \rangle = \langle \bar{a} \rangle + \langle \bar{b} \rangle$: each party P_i computes the addition of its shares $\langle \bar{c}_i \rangle \leftarrow \langle \bar{a}_i \rangle + \langle \bar{b}_i \rangle$ and the addition of the MACs on the secrets $\gamma(c)_i = \gamma(a)_i + \gamma(b)_i$. We can verify that $\alpha \cdot c = \sum \gamma(c)_i = \sum (\gamma(a)_i + \gamma(b)_i) = \alpha \cdot (a + b)$. To check the commitment of everybody to its share, we have to check if indeed $\gamma = \alpha a$. We can do this avoiding to open α as it is the long-term key. It suffices to observe that the value $\gamma - \alpha a$ is a linear function of shared values γ and α which means that players can locally compute shares and then check whether it is equal to 0. If this is not the case, the protocol immediately halts. As multiplication using multiplication triples rests on the opening of additions of secret shares and other additions, we can implement this scheme into the multiplication as well [28].

The use of MACs to commit the parties to their shares makes the protocol secure in the *active case.*

3.2.5 The SCALE-MAMBA framework

Different implementations of multi-party computation protocols exist. For frameworks based on bit-wise decomposition and circuit garbling, both Fairplay [51] and BMR [11] are multi-party. Some hybrid frameworks using bit-wise decomposition as well as secret sharing protocols also exist such as SEPIA [19] or Rmind [16], but are limited to certain pre-computed functions executions and do not allow real programming. General frameworks that allow programming include VIFF [27] (which is obsolete as it has not been updated anymore since 2009) and Sharemind [15], [18] that uses its own programming language SecreC, based on the C-language [17], [45], [67] and that is proprietary and fixed to three parties. ABY is fixed to two parties [30]. The sole framework that is open-source and n-party is SPDZ [28], [29], which is recently discontinued and taken over by the SCALE-MAMBA project [4]. This is the one used in this thesis.

SCALE-MAMBA (short for Secure Compuation Algorithms from LEuven and Multiparty AlorithMs Basic Argot) also incorporates some other works ([12], [60]) and has developed its own language based on Python (MAMBA). A major difference with SPDZ is that is thought as of an entire system and not a set a functions that can be used, but that the programmer still has to compose together. It is therefore not suited for benchmarks on specific protocols, but very well suited for the benchmarking of general algorithms: the goal of this thesis. The implementations in this thesis are using a 128-bits prime p with $k = 64$ and statistical security parameter $\kappa = 40$.

SCALE-MAMBA also has a native support for high-level multi-thread protocols where the programmer should not care about shared variables, there is only a protocol for starting the different threads and another one for waiting that all threads are done.

Furthermore, multi-threading is at the core of SCALE-MAMBA as it natively uses four continuous separate "feeder" threads for the offline phase, one for producing the multiplication triples, another for the square pairs, a third for shared bits and the last for other various operations such as the sacrificing[2]. This use of "feeder" threads allow to continuously produce the elements needed for the online phase without having to match both of them, which appears to be a complex enterprise. Additional global threads are also used to produce the cyphertexts used for the FHE component. However, the user has no control on all these threads as the are built into the core of the framework [4].

[2]The multiplication triples are continuously produced but need to be sacrificed before consumption by the online stage. This thread continuously sacrifices triples and add them to a sacrificed-list, ready for use.

3.2.6 Performance indicators

When evaluating the algorithm, we should be careful at four performance indicators: the round cost, the computational cost, the communication cost and the classification performance. As protocols run online, we should also monitor the consequences on the network usage as well as the costs on each machine. At last the real performance of the classification itself should not be forgotten.

Round cost

The round cost is the number of rounds performed in total and is a measure of the network usage. As the protocol has to wait for the exchange of every shares before going further — waiting that a round finishes —, a high round number means a high dependence on the latency of the network. Indeed, if the network is slow, the exchanges will be the bottleneck of the protocol and a high number of rounds will by synonymous with a low execution.

Computational cost

The computational cost is a measure of the cost on local machine, for each party. This is measured as the amount of time each CPU takes to execute its part of the protocol. Computational cost seems less studied than other indicators when using MPC but is key to a rapid execution and to measure the practicality of an algorithm [14]. As we aim to measure the practicality of algorithms in real-case scenarios, this indicator should absolutely not be neglected.

Communication cost

The communication cost indicator measures the quantity of data transmitted over the network. This can be a bottleneck when the communication on the network is very slow and has a small bandwidth. For a round to be completed, all the data has to be transmitted. The communication cost is therefore closely related to the round cost.

Classification performance

This has nothing to do with the multi-party computation in itself but is crucial to a working system. This whole thesis aims at finding the trade-offs to improve the practical feasibility of privacy-friendly machine-learning algorithms. The trade-off between the classification performance and the MPC-feasibility is surely the most important. An very fast MPC-based algorithm is not relevant if it does not do its job correctly enough. The opposite is less true: a very slow machine-learning algorithm is — it is true not so interesting — but still relevant when it classifies the data correctly. Of course, the real goal is to find one that satisfies both criteria.

3.3 Protocols implemented

As said before, we aim here at evaluating a data query of a first party against the data-base of a second party using various machine-learning algorithms. The goal here is to preserve the privacy of the data of both parties: the query and the model parameters. In the following sections, $\langle a \rangle_i$ and $\langle a \rangle_{ij}$ denote a secret in matrix (or array) of secrets and should not be confused with $\langle a_i \rangle$ which denote a share of Secure in the previous sections. $\langle A \rangle$ denotes the corresponding matrix. Similarly, we use the notation $\langle A \rangle_{i:}$ or $\langle A \rangle_{:j}$ to denote the vector corresponding to the i-th line or j-th column of the matrix of secret values.

3.3.1 Principal component analysis

Let us consider the matrix of the PCA weights W_{n_{pca},n_f} and a list of queries to be transformed X_{n_q,n_f} were n_{pca} the number of principal components, n_q is the number of queries and n_f the number of features. Transforming these queries into the principal components scores $T_{n_q,n_{pca}}$ described in W just corresponds to the matrix multiplication operation $T = X \cdot W^T$. This operation is securely implemented in algorithm 10.

As we can see, this simple matrix multiplication has the theoretical computational complexity of the naive matrix multiplication algorithm $\mathcal{O}(n_q n_{pca} n_f)$, which is cubic.

There is here total secrecy on the values of the queries and the PCA weights. The only public information are the indices of the loops, but inside the loops, a totally opaque operation occurs. To summarize, the secrecy is preserved for all the data except its sizes: the number of principal components kept n_{pca}, the number of queries n_q and the number of features n_f.

```
Data: $n_{pca} \times n_f$ matrix of principal components weights $\langle \tilde{W} \rangle$ and $n_q \times n_f$ matrix
      of queries $\langle \tilde{X} \rangle$.
Result: $n_q \times n_{pca}$ matrix of the $n_{pca}$ first components scores of each query $\langle \tilde{T} \rangle$.
initialize $\langle \tilde{T} \rangle$ of size $n_q \times n_{pca}$
foreach $i = 1, \ldots, n_q$ do
    foreach $j = 1, \ldots, n_{pca}$ do
        foreach $k = 1, \ldots, n_f$ do
            $\langle \tilde{t} \rangle_{ij} \leftarrow \langle \tilde{t} \rangle_{ij} + \langle \tilde{w} \rangle_{jk} \cdot \langle \tilde{x} \rangle_{ik}$
        end
    end
end
return $\langle \tilde{T} \rangle$
```

Algorithm 10: Secure PCA evaluation protocol.

3.3.2 Linear support vector machines

The linear support vector machine protocol is basically the same as the PCA evaluation protocol as it also consists in a matrix multiplication, with the difference that an intercept vector has to be added. This is the vector containing the bias terms of the different SVMs. The protocol evaluates a linear support vector machine in its primal form which consists of a scalar product that sees a constant added to it. However, as we work with multi-class support vector machines, we have to consider multiple scalar products and similarly, as we work with more than one query, this corresponds to a matrix multiplication in total.

Let us consider now a matrix W_{n_c,n_f} where n_c is the number of classes — each SVM represents one class — that contains the primal weights of the linear support vector machine. To that we have to add the intercept vector B_{n_c}. The list of queries is the same as before X_{n_q,n_f}. The output is a matrix containing the score of for each query on each SVM S_{n_c,n_q}. The operation becomes $S = WX^T + B$. The protocol implementing it is given at algorithm 11.

The complexity is similar as before: $\mathcal{O}(n_c n_q n_f)$. If we use feature space reduction and the queries are first transformed using the principal component analysis (n_f is reduced to n_{pca}), it becomes $\mathcal{O}(n_c n_q n_{pca})$. However, we eventually have to add the complexity of the PCA transformation itself, which gives a total of $\mathcal{O}(n_{pca} n_q (n_f + n_c)) > \mathcal{O}(n_c n_q n_f)$ for $n_f, n_c \geq 1$ (which is always verified). Hence, we should not expect the linear support vector machines to be more effective using a PCA feature reduction. However, this is not the case for the χ^2-feature reduction linear support vector machine where the complexity is always slower than the full-feature linear support vector machine. Indeed, we have $\mathcal{O}(n_c n_q n_{\chi^2}) < \mathcal{O}(n_c n_q n_f)$ if $n_{\chi^2} < n_f$ (which should be the case for a working χ^2 feature selection).

Here again, the secrecy is preserved in all the data except the sizes, adding here the size n_c of the χ^2 features kept to the ones already discussed in the subsection above. We could decide to hide n_c by just using a null SVM weight for a coefficient that has been dropped by the selection, but this would not allow to reduce the complexity. This is of no interest as the whole idea of χ^2 feature selection is to increase the evaluation speed.

```
Data: $n_c \times n_f$ matrix of the $n_c$ different SVM weights $\langle\tilde{W}\rangle$, $n_q \times n_f$ matrix of
      queries $\langle\tilde{X}\rangle$ and an intercept vector $\langle\tilde{B}\rangle$ of length $n_c$.
Result: $n_c \times n_q$ matrix containing the $n_c$ SVM scores for each query $\langle\tilde{S}\rangle$.
initialize $\langle\tilde{T}\rangle$ of size $n_q \times n_{pca}$
foreach $i = 1, \dots, n_q$ do
    foreach $j = 1, \dots, n_c$ do
        foreach $k = 1, \dots, n_f$ do
            $\langle\tilde{s}\rangle_{ji} \leftarrow \langle\tilde{s}\rangle_{ji} + \langle\tilde{w}\rangle_{jk} \cdot \langle\tilde{x}\rangle_{ik}$
        end
        $\langle\tilde{s}\rangle_{ji} \leftarrow \langle\tilde{s}\rangle_{ji} + \langle\tilde{b}\rangle_{j}$
    end
end
return $\langle\tilde{S}\rangle$
```

Algorithm 11: Secure linear SVM evaluation protocol.

3.3.3 Support vector machines with radial based functions

As a reminder the evaluation of a query x on linear support vector using a kernel functions is only possible in the dual space and is given by

$$\texttt{SVM}(x) = \sum_{i=1}^{n_{svm}} \alpha_i y_i K(x_i, x) + b, \tag{3.5}$$

where x_i are the n_{svm} support vectors x_i and $K(x_i, x)$ the kernel function here given by

$$K\left(x_i, x_j\right) = \exp\left(-\frac{\left\|x_i - x_j\right\|^2}{2\sigma^2}\right), \tag{3.6}$$

with σ^2 a kernel parameter. Unfortunately, SCALE-MAMBA has no support for exponentiation of secret fixed point exponentiation. This can be easily circumvented using a Taylor expansion of the exponential function

$$e^x \approx \sum_{n=0}^{n_e} \frac{x^n}{n!}, \tag{3.7}$$

with n_e the number of iterations we want to perform (the more the better). The secure radial based kernel function protocol is developed in algorithm 13 using the secure distance computation protocol described in algorithm 12. Once these protocols are defined, we can use the protocol described at algorithm 14 to compute the transformation of the queries into the dual space, which is equivalent to the kernel matrix between the support vectors and the queries. These transformed queries can then be used with the classical linear support vector machines, but with the dual α_i coefficients instead of the primal weights which changes the n_f estimations to n_{sv} in the protocol described at algorithm 11. Though, one dual transformation is needed per different SVM.

Let's now have a look at the complexity. The complexity of the dual transformation is $\mathfrak{O}(n_f n_e n_{svm})$ per SVM. As there are n_{svm} different SVMs, this gives $\mathfrak{O}(n_f n_e n_{svm}^2)$ for all the transformations. If we add the estimation of the SVMs themselves based on these dual transformations, this gives a total of $\mathfrak{O}(n_{svm}^2(n_q + n_e n_f))$. The use of support vectors in the dual makes the complexity of degree 4 and not cubic anymore. The next chapter aims at investigating if this trade-off is worth the cost.

Similarly as before, we can also use a PCA reduction before the computation of the support vectors and try to gain speed on the distance computation. As we saw before, the PCA reduction has a cubic complexity and this should thus not change the degree of the estimation, but reduce its factors to $\mathfrak{O}(n_{svm}^2(n_q + n_e n_{pca}))$ where n_{pca} is the number of components kept. Computationally speaking, this trade-off here is expected to deliver better results. Using χ^2 selection reduces the complexity to $\mathfrak{O}(n_{svm}^2(n_q + n_e n_{\chi^2}))$ which should also deliver better results. The choice between PCA and χ^2 feature selection was trivially favouring the latter for the linear SVM in every case, here it tends to favor the first one if $n_{pca} < n_{\chi^2}$ as the computation PCA evaluation has no significant influence on the complexity here, which was not the case here above.

```
Data: secret vector $\langle\tilde{U}\rangle$ of size $n_f$ and secret vector $\langle\tilde{V}\rangle$ of same size.
Result: $\langle\tilde{d}\rangle$ corresponding to $d = \|U - V\|^2$.
foreach $i = 1, \ldots, n_f$ do
    $\langle\tilde{d}\rangle \leftarrow \langle\tilde{d}\rangle + (\langle\tilde{u}\rangle_i - \langle\tilde{v}\rangle_i)^2$
end
return $\langle\tilde{d}\rangle$
```

Algorithm 12: Secure distance computation protocol `Dist`.

```
Data: secret vector $\langle\tilde{U}\rangle$ of size $n_f$, secret vector $\langle\tilde{V}\rangle$ of same size and secret
        kernel parameter $\langle\tilde{a}\rangle$ sharing $2\sigma^2$.
Result: $\langle\tilde{d}\rangle$ corresponding to $K(u, v)$ with parameter $2\sigma^2$.
$\langle\tilde{v}\rangle \leftarrow -\texttt{Dist}(\langle\tilde{U}\rangle, \langle\tilde{V}\rangle)/\langle\tilde{a}\rangle$
foreach $i = 0, \ldots, n_e$ do
    $\langle\tilde{d}\rangle \leftarrow \langle\tilde{d}\rangle + \langle\tilde{v}\rangle^i/i!$
end
return $\langle\tilde{d}\rangle$
```

Algorithm 13: Secure radial based kernel function evaluation protocol `Kernel`.

```
Data: secret matrix of queries $\langle\tilde{X}\rangle$ of size $n_q \times n_f$, secret matrix of support
        vectors $\langle\tilde{S}\rangle$ of size $n_{svm} \times n_f$ and secret kernel parameter $\langle\tilde{a}\rangle$.
Result: $\langle\tilde{T}\rangle$ transformed queries into the dual space.
initialize $\langle\tilde{T}\rangle$ of size $n_q \times n_{svm}$
foreach $i = 1, \ldots, n_q$ do
    foreach $j = 1, \ldots, n_{svm}$ do
        $\langle\tilde{t}\rangle_{ij} \leftarrow \texttt{Kernel}(\langle\tilde{X}\rangle_{i:}, \langle\tilde{S}\rangle_{j:}, \langle\tilde{a}\rangle)$
    end
end
return $\langle\tilde{T}\rangle$
```

Algorithm 14: Secure dual space transformation protocol.

3.3.4 Multi-class evaluation

The SVMs only perform binary classification and are combined to perform multi-class classification. To evaluate the one-against-all model, we just have to take the maximum score of each SVM and return the corresponding target. In the protocol described in algorithm 15, the final score of the wining SVM is not returned and sole the corresponding class is. In the following algorithms, the notation $\langle -\tilde{\infty}\rangle$ describes a very small value that is certainly smaller than all SVMs scores (these are typically comprised between -1 and 1 and a value of -1000 suffices). In the very unlikely case where no score is bigger than this value, the algorithm will return Secure index 0. The complexity of the algorithm is equal to $\mathcal{O}(n_q n_c)$ with n_c the number of classes, but incorporates for the first time comparisons.

Similarly, we can define the algorithm picking the winner for a tree-based model. This is done by picking the index of the highest SVM in the tree returning a positive score. Once a SVM returns a positive score, all next scores become not relevant. However, to preserve the security, we still need to evaluate all SVMs and cannot stop even if the SVM higher in the tree returned one. If we did this, an attacker could just look at the index of the SVM when the algorithm stops to disclose the winning target. The protocol for defining the winning target in the case of a tree-based multi-class model is given at algorithm 16. In this case, it is not possible that the algorithm returns 0 and finds no winner. The computational complexity is exactly the same as the one-against-all, but $n_c = 4$ and not to 5.

```
Data: $n_c \times n_q$ matrix containing the $n_c$ SVM scores for each query $\langle\tilde{S}\rangle$ and an
        integer vector corresponding to index of the classes $\langle\bar{C}\rangle$ of size $n_c$.
Result: vector $\langle\bar{N}\rangle$ of size $n_q$ containing the winning indices for each query.
initialize vector $\langle\bar{N}\rangle$ of size $n_q$ to $\langle\bar{0}\rangle$
foreach $i = 1, \dots, n_q$ do
    $\langle\tilde{m}\rangle \leftarrow \langle\tilde{\infty}\rangle$
    foreach $j = 1, \dots, n_c$ do
        $\langle\bar{b}\rangle \leftarrow \langle\tilde{S}\rangle_{ji} > m$
        $\langle\tilde{m}\rangle \leftarrow_{\langle\bar{b}\rangle} \langle\tilde{S}\rangle_{ji} : m$
        $\langle\bar{N}\rangle_i \leftarrow_{\langle\bar{b}\rangle} \langle\bar{C}\rangle_j : n$
    end
end
return $\langle\bar{N}\rangle$
```

Algorithm 15: Secure one-against-all multi-class evaluation protocol.

```
Data: $n_c \times n_q$ matrix containing the $n_c$ SVM scores for each query $\langle\tilde{S}\rangle$ and an
        integer vector corresponding to index of the classes $\langle\bar{C}\rangle$ of size $n_c$.
Result: vector $\langle\bar{N}\rangle$ of size $n_q$ containing the winning indices for each query.
initialize vector $\langle\bar{N}\rangle$ of size $n_q$
foreach $i = 1, \dots, n_q$ do
    foreach $j = n_c, \dots, 1$ do
        $\langle\bar{b}\rangle \leftarrow \langle\tilde{S}\rangle_{ji} > 0$
        $\langle\bar{N}\rangle_i \leftarrow_{\langle\bar{b}\rangle} \langle\bar{C}\rangle_j : n$
    end
end
return $\langle\bar{N}\rangle$
```

Algorithm 16: Secure tree-based multi-class evaluation protocol.

3.3.5 Nearest neighbours

The secure evaluation of the k-NN protocol happens in two phases, the first one is computing the distances between the query and the different points of the training size n_t. This is done using the protocol described in algorithm 12. The points are then ordered and the winning target is returned. Here again, the smallest distance is not returned to avoid attacks where someone would send a lot of queries of data-points near a certain secret point of the training data-set and try to triangulate its position[3].

There are different possibilities for implementing the algorithm (we use a minimum selection algorithm with complexity $\mathcal{O}(n \log n)$:

- The first idea is to compute all distances which has a complexity $\mathcal{O}\left(n_t n_f\right)$ and then go k times through those distances to find the k smallest ones, which has complexity $\mathcal{O}\left(l n_t \log n_t\right)$. This algorithm, which calculates the distances, stores them and go k times through them has total complexity $\mathcal{O}\left(n_t n_f + k n_t \log n_t\right)$, but requires $\mathcal{O}\left(n_t\right)$ extra memory.
- An alternative could be to avoid the storing of the distances and compute them during the classifying. The algorithm goes k through all observations as before, but computes the distance at that moment. The advantage is the requirement of no extra memory, however the computational complexity becomes $\mathcal{O}\left(n_f k n_t \log n_t\right)$.
- A third alternative could be to use Hoare's *quickselect* algorithm [40]. This algorithm is able to find the k^{th} smallest — alternatively biggest — elements of a list in best case scenario $\mathcal{O}\left(n_t\right)$ and worst case scenario $\mathcal{O}\left(n_t \log n_t\right)$. The practice shows that the algorithm tends to obtain a complexity closer to the best case than the worst case scenario. Using this algorithm, one could hope to obtain a k-NN computational complexity of $\mathcal{O}\left(n_t n_f\right)$. Unfortunately, this algorithm uses recursive function calls which are currently not supported in the SCALE-MAMBA framework. More discussion is given in the next sections and in appendix C. As the indices are not secret, this would also reveal part of the sorting pattern, but as the distances are not revealed, triangulation-based attacks would still not be possible.

As storing is not really a concern in secret sharing, but the bottleneck is the computation part. The first option has been chosen. The secure k-NN evaluation protocol is given at algorithm 17 for a unique query. One just has to rerun it n_q times to compute multiple queries. This protocol rests on two other sub-protocols, the first one aims at defining the majority class present in the neighbours and is described at algorithm 18. It first consists in a counting of the elements present in the neighbours and searching for the maximum. In case of tie, the highest value tied class is returned. This choice is made to limit the number of false negatives[4].

When $k = 1$, the part of the algorithm where the majority voting is used can be dropped as well as the need for selecting a minimum higher to the one previously

[3]As we are working in a feature space of dimension d, $d + 1$ points are still needed for triangulation in the favorable case where all these points are the closest to the targeted point and return their distance to it and not another closer point.

[4]This depends on the data-set used in the next chapter. In that data-set the higher classes tend to be more difficultly recognizable. This case is also to be nuanced as the algorithm as we will see performs the best for $k = 1$ where no ties are possible.

found (which comprises two time more comparisons). For this purpose, two protocols for the computation of a minimum are given. One for the $k = 1$ case at algorithm 19 and another for $k > 1$ at algorithm 20. The reason we have to use a specific protocol for the selection of an minimum when a threshold (the previous minimum for $k - 1$) is because it is not possible to modify the current distance array to change the value of the previously found distance or just to suppress that element without revealing its index. Both protocols have complexity $\mathcal{O}(n_t \log n_t)$. For the sake of simplicity, they are referred to by the same name.

The complexity of the protocol for n_q queries is $\mathcal{O}(n_q\{n_t n_f + k n_t \log n_t\})$, $\mathcal{O}(n_q\{n_{pca}(n_t + n_f) + k n_t \log n_t\})$ with PCA feature reduction and $\mathcal{O}(n_q\{n_t n_{\chi^2} + k n_t \log n_t\})$ with χ^2 feature selection.

```
**Data**: query vector $\langle\tilde{X}\rangle$ of size $n_f$, matrix of training points $\langle\tilde{P}\rangle$ of size $n_t \times n_f$
      and corresponding classes vector of $\langle\bar{C}\rangle$ of size $n_t$.
**Result**: class $\langle\bar{l}\rangle$ for the query.
initialize vector $\langle\tilde{D}\rangle$ of size $n_q$
**foreach** $i = 1, \dots, n_t$ **do**
    $\langle\tilde{d}\rangle_i \leftarrow$ Dist($\langle\tilde{X}\rangle, \langle\tilde{P}\rangle_{i:}$)
**end**
$\langle\tilde{m}\rangle \leftarrow \langle\bar{-\infty}\rangle$
initialize vector $\langle\bar{T}\rangle$ of size $k$
**foreach** $i = 1, \dots, k$ **do**
    $\langle\bar{t}\rangle_i, \langle\tilde{m}\rangle \leftarrow$ Min($\langle\tilde{D}\rangle, \langle\bar{C}\rangle, \langle\tilde{m}\rangle$)
**end**
$\langle\bar{l}\rangle \leftarrow$ MajVote($\langle\bar{T}\rangle$)
**return** $\langle\bar{l}\rangle$
```

Algorithm 17: Secure k-NN evaluation protocol.

```
Data: vector $\langle\bar{T}\rangle$ containing the targets of the $k$ nearest neighbours.
Data: $\langle\bar{l}\rangle$ containing the most present target. In case of a tie, the highest tied
      class is returned.
initialize vector $\langle\bar{r}\rangle$ of size $n_c$ to 0
foreach $i = 1, \ldots, n_c$ do
    foreach $j = 1, \ldots, k$ do
        $\langle\bar{b}\rangle \leftarrow \langle\bar{t}\rangle_j = i$
        $\langle r\rangle_i \leftarrow_{\langle\bar{b}\rangle} \langle r\rangle_i + 1 : \langle r\rangle_i$
    end
end
$\langle\bar{l}\rangle \leftarrow 0$
$\langle\bar{c}\rangle \leftarrow 0$
foreach $i = 1, \ldots, n_c$ do
    $\langle\bar{b}\rangle \leftarrow \langle r\rangle_i \geq \langle\bar{y}\rangle$
    $\langle\bar{c}\rangle \leftarrow_{\langle\bar{b}\rangle} \langle r\rangle_i : c$
    $\langle\bar{l}\rangle \leftarrow_{\langle\bar{b}\rangle} i : l$
end
return $\langle\bar{l}\rangle$
```

Algorithm 18: Secure majority vote evaluation protocol `MajVote`.

```
Data: vector of distances $\langle\tilde{D}\rangle$ of size $n_t$ and corresponding classes $\langle\bar{C}\rangle$.
Result: shortest distance $\langle\tilde{d}\rangle$ and corresponding class $\langle\bar{c}\rangle$.
if $n_t = 1$ then
    return $\langle\tilde{d}\rangle_1, \langle c\rangle_1$
end
else
    $n' \leftarrow \lfloor (n_t + 1)/2 \rfloor$
    initialize $\langle\tilde{D}'\rangle$ of size $n'$
    initialize $\langle\bar{C}'\rangle$ of size $n'$
    foreach $i = 1, \ldots, \lfloor n_t/2 \rfloor$ do
        $\langle\bar{b}\rangle \leftarrow \langle\tilde{d}\rangle_{2i} < \langle\tilde{d}\rangle_{2i+1}$
        $\langle\tilde{d}'\rangle_i \leftarrow_{\langle\bar{b}\rangle} \langle\tilde{d}\rangle_{2i} : \langle\tilde{d}\rangle_{2i+1}$
        $\langle\tilde{c}'\rangle_i \leftarrow_{\langle\bar{b}\rangle} \langle\tilde{c}\rangle_{2i} : \langle\tilde{c}\rangle_{2i+1}$
    end
    if $n \mod 2 = 1$ then
        $\langle\tilde{d}'\rangle_{n'-1} \leftarrow \langle\tilde{d}\rangle_{n_t-1}$
        $\langle\tilde{c}'\rangle_{n'-1} \leftarrow \langle\tilde{c}\rangle_{n_t-1}$
    end
    return Min($\langle\tilde{D}'\rangle, \langle\tilde{C}'\rangle$)
end
```

Algorithm 19: Secure minimum evaluation protocol `Min` ($k = 1$).

```
Data: vector of distances $\langle \tilde{D} \rangle$ of size $n_t$, corresponding classes $\langle \bar{C} \rangle$ and
      minimum threshold $\langle \tilde{m} \rangle$.
Result: shortest distance $\langle \tilde{d} \rangle$ and corresponding class $\langle \bar{c} \rangle$.
if $n_t = 1$ then
    return $\langle \tilde{d} \rangle_1, \langle c \rangle_1$
end
else
    $n' \leftarrow \lfloor (n_t + 1)/2 \rfloor$
    initialize $\langle \tilde{D'} \rangle$ of size $n'$
    initialize $\langle \bar{C'} \rangle$ of size $n'$
    foreach $i = 1, \ldots, \lfloor n_t/2 \rfloor$ do
        $\langle \bar{b} \rangle \leftarrow \langle \tilde{d} \rangle_{2i} < \langle \tilde{d} \rangle_{2i+1}$
        $\langle \tilde{d''} \rangle \leftarrow_{\langle \bar{b} \rangle} \langle \tilde{d} \rangle_{2i} : \langle \tilde{d} \rangle_{2i+1}$
        $\langle \tilde{c''} \rangle \leftarrow_{\langle \bar{b} \rangle} \langle \tilde{c} \rangle_{2i} : \langle \tilde{c} \rangle_{2i+1}$
        $\langle \tilde{d'''} \rangle \leftarrow_{\langle \bar{b} \rangle} \langle \tilde{d} \rangle_{2i+1} : \langle \tilde{d} \rangle_{2i}$
        $\langle \tilde{c'''} \rangle \leftarrow_{\langle \bar{b} \rangle} \langle \tilde{c} \rangle_{2i+1} : \langle \tilde{c} \rangle_{2i}$
        $\langle \bar{b'} \rangle \leftarrow \langle \tilde{d''} \rangle > \langle \tilde{m} \rangle$
        $\langle \tilde{d'} \rangle_i \leftarrow_{\langle \bar{b'} \rangle} \langle \tilde{d''} \rangle : \langle \tilde{d'''} \rangle$
        $\langle \tilde{c'} \rangle_i \leftarrow_{\langle \bar{b'} \rangle} \langle \tilde{c''} \rangle : \langle \tilde{c'''} \rangle$
    end
    if $n \mod 2 = 1$ then
        $\langle \tilde{d'} \rangle_{n'-1} \leftarrow \langle \tilde{d} \rangle_{n_t-1}$
        $\langle \tilde{c'} \rangle_{n'-1} \leftarrow \langle \tilde{c} \rangle_{n_t-1}$
    end
    return Min($\langle \tilde{D'} \rangle, \langle \tilde{C'} \rangle$)
end
```

Algorithm 20: Secure minimum evaluation protocol with threshold `Min` ($k > 1$).

Chapter

4 Experimental results and discussion

4.1 Methodology

This chapter aims at evaluating the different algorithms and the possible feature and instance set reductions possible to make them suitable for a practically feasible secure usage. Each algorithm is first tested with all data reduction methods to assess which trade-offs can be made on the classification performance. Finally, they are tested in a MPC setting to observe how the reductions reduce their computational, round and communication costs.

To investigate the classification performance, we use different indicators based on the classification *confusion matrix*, also called *error matrix*. This matrix represents the results of the classification. Each column stands for the number of occurrences of a real class, whereas the lines stand for the observed classes. If the classifier is perfect, all estimated classes should correspond to the real ones and the classification matrix should only have non-zero elements on its diagonal. In the specific case of binary classifiers (class/non-class), the elements of the classification matrix often are referred to as true positives (f_p), true negatives (t_n), false positives (f_p) and false negatives (f_n):

$$\begin{pmatrix} t_p & f_n \\ f_p & t_n \end{pmatrix}. \tag{4.1}$$

Still, we would like to have some classification performance indicators that are one-dimensional for the ease of the comparison of the different methods. From the confusion matrix, we can build different classifiers. Of course, it is never possible to build a perfect indicator as the reduction of the confusion matrix to one dimension is inevitably accompanied by some information loss. The first one is the accuracy that represents the proportion of well-classified elements

$$\texttt{Accuracy} = \frac{t_p + t_n}{t_p + t_n + f_p + f_n}, \tag{4.2}$$

and can be generalized in multi-class systems (the trace of the confusion matrix divided by its total number of elements). It's maximum is always 1 and minimum always 0. The accuracy indicator has a few limitations and the most notable one is its lack of sensibility to the initial distribution. Let's imagine a binary system where the first class is very well classified and comprises the majority of the proportion of the test set. The other class is highly misclassified but represents a very limited proportion of the test set. The accuracy would be high as most of the data (the first class) is well classified, though the other class is not and almost doesn't count in the accuracy indicator, whatever its performance.

This problem can be solved by considering *Matthew's correlation coefficient* [52] that is defined as

$$\mathtt{MCC} = \frac{t_p \times t_n - f_p \times f_n}{\sqrt{(t_p + f_p)(t_p + f_n)(t_n + f_p)(t_n + f_n)}}. \tag{4.3}$$

This indicator can be interpreted as the correlation between predictions and observations, hence its name. For multi-class models, the MCC will be computed as the mean for all the classes. This makes sense as the indicator doesn't depend on the order of the class (e.g. in the binary case: the indicator is the same whatever is considered as the positive or negative class). Its value always range from -1 for the worst case to 1 forst the best case.

A last indicator will be considered: *Cohen's kappa coefficient* measures the difference between the measured classifier and a random classifier. More specifically, it is given by

$$\mathtt{Kappa} = \frac{\mathtt{Accuracy} - p_e}{1 - p_e}, \tag{4.4}$$

where p_e represents the accuracy of a random classifier and is given by

$$p_e = \frac{p(t_p + f_p) + n(f_n + t_n)}{(t_p + t_n + f_p + f_n)^2}, \tag{4.5}$$

where p and n are the proportion of positive and negatives occurrences. For multi-class models, it can be generalized by considering t_p (resp. t_n, f_p and f_n) as the sum of the same t_p (resp. t_n, f_p and f_n) for each individual binary classifier [25]. Here again, it gives an answer to the problem of unequal classes distributions. In a binary system where a class is much more present than the other, the value for p_e will be high as the random classifier will assign each element with a high probability of being the much more present class. In the extreme case of all the elements being in the same class, the random classifier will assign each element to that only class. Its respective probability would be 1 and Kappa 0. Though, the accuracy would be 100%.

All experiments were done on a 3,3GHz Intel Core i7 dual-core (Skylake) processor with 16GB RAM computer. The machine-learning classification performance experiments have been done using MATLAB R2018a.

4.2 The NSL-KDD data-set

For the benchmark and testing of the model, the NSL-KDD data-set was used [31], [85]. This data-set is an enhanced version of the KDDCUP'99 [32], [82] data-set originally built upon 4GB of compressed raw traffic data captured during 7 weeks. The KDDCUP'99 extracted a little bit less than 5 million single connections as vectors of 41 different features. It has then been the most used data-set for evaluation of intrusion detection systems. However, it had a few issues such as a lot of redundancy. These issues were solved in the new data-set it was based upon: the NSL-KDD data-set.

4.2.1 Attack classes

Network attacks detected by intrusion detection systems are highly diversified. In total, about 40 different attack classes exist, but are often regrouped into four main classes

- **Normal (67,343 samples)**: This just corresponds to a non-attack, a normal connection.
- **DoS (45,927 samples)**: This is a denial of service attack, where the attacker makes the some resources too busy to be able to treat legitimate requests.
- **Probe (11,656 samples)**: This is a probing attack where the attacker tries to gain an understanding of the remote victim like the port it is using or the structure of its network.
- **U2R (52 samples)**: In this class, the attacker try to gain an unauthorized access to the root user account starting with a normal user account. These attacks are much more complex as the attacker has to exploit some vulnerabilities of the system. By consequence, they are also a lot less common. The difficulty in this data-set is the little amount of samples of this attack class.
- **R2L (995 samples)**: Here, the attacker tries to gain unauthorized local access from a remote machine. Here again, this attack demands more complex attacks than probing or denial of service, such as sniffing or social engineering. This class suffers thus from the same detection problem as the U2R class, but in a lesser manner.

4.2.2 Features and pre-processing

The machine-learning algorithms described in chapter 2 all work with data-points in a hyper-space, meaning that all features have to be numeric values. However, in addition to the output class, 3 of the 41 features are categorical and not numeric. We therefore need to convert these values to numeric ones.

The first idea it to replace the feature by n new binary features corresponding to the n different categories. However, we must also be careful not augmenting too much the length of the vector to avoid what is called the *curse of dimensionality*. These are problems appearing in high dimensionality hyper-spaces that are not appearing in lower dimensions: as the training set size is fixed, the data-points become more and more isolated as the feature dimension augments. This induces more and more sparsity in the training data-set.

As we discussed before, new dimensions may also imply more parameters in the final model. This implies more complexity allowed to the models and a greater risk of overfitting. The goal is in fact more to reduce the feature-space size than augmenting it. The first of the three categorical features describes the protocol used by the connection and has a total number of only three different categories. We can thus use this technique and just augmenting the total feature vector dimension from 41 to 43.

The two other categorical feature have respectively 70 and 11 different categories. Using the same method for both would increase the dimension from 43 to 122, a totally unreasonable choice considering the curse of dimensionality and the risk of overfitting. We therefore have to find a method to convert categorical data into a one-dimensional value. In the case of intrusion detection systems, Aburoman et al. have simply mapped the categories to a numeric value ranging from 1 to n [1]. However, as many machine-learning algorithms — and all of the ones used in this thesis and the ones they used in their work — are based on similarity, measured by distance, this would arbitrarily make two categories similar to each other or not based on the order of the mapping. Conversion using the ASCII mapping has also been used with intrusion detection systems [81]. To still give similarity a coherent meaning after the mapping, we used a frequency conversion: each category is replaced by its frequency in the data-set. Of course, the information is lost, but this is inevitable when reducing dimension.

The feature vector also exists out of a lot of binary values. These can just be considered numeric as such. Table 4.1 gives a list of the different features after conversion. As can be seen, the different features are quite differing in scales which can be a problem when computing distances for example: a certain feature will take all the weight making the other values non-relevant. We use a simple linear scaling to the interval $[-1, 1]$:

$$x' = 2\frac{x - \mathtt{min}(x)}{\mathtt{max}(x) - \mathtt{min}(x)} - 1. \tag{4.6}$$

We can now hope that all features have the same weight. In this thesis, we will not consider secure normalization, as its main goal is to bring all features to a similar scale. The exact values are not very relevant as long as every party uses the same ones. Normalization can be considered as part of the feature extraction if everyone agrees on common maximum and minimum values.

#	Name	Description	Type	Maximum
1	connection_length	Length of time of the connection	N	42908
2	protocol_udp	UDP protocol used in the connection	C	1
3	protocol_tcp	TCP protocol used in the connection	C	1
4	protocol_icmp	ICMP protocol used in the connection	C	1

continued on next page

Table 4.1 *continued from previous page*

#	Name	Description	Type	Maximum
5	network_service	Destination network service used for the connection	C	N/A
6	status_flag	Connection status flag	C	N/A
7	data_size_in	Number of bytes transferred in single connection from source to destination	N	1379963888
8	data_size_return	Number of bytes transferred in single connection from destination to source	N	1309937401
9	IP_check	checks whether IP adresses and port from source and destination are equal	B	1
10	wrong_fragments	Number of wrong fragments in the connection	N	3
11	number_urgent_bits	Number of packets with "urgent" bit activated in the connection	N	3
12	number_hots	Number of "hot" indicators in the content (eg. access to directory, creation or execution of programs)	N	77
13	count_failed	Count of unsuccessful login attempts	N	5
14	login_success	Indicator of successful login	B	1
15	number_compromised	Number of compromised conditions	N	7479
16	root_flag	Indicator of "root" shell has been obtained	B	1
17	number_su	Number of attempts of "su root" command	N	2
18	number_root_ops	Number of operations or accesses preformed as "root" connection	N	7468
19	number_new_files	Number of new files in the connection	N	43

continued on next page

Table 4.1 *continued from previous page*

#	Name	Description	Type	Maximum
20	number_shells_prompt	Number of shell prompts in the connection	N	2
21	number_access_control	Number of "access control" files operations	N	9
22	number_outbnd_cmd	Number of outbound commands in an ftp connection	N	0
23	hot_login_flag	Indicator of login user on "hot" list (root or admin)	B	1
24	guest_flag	Indicator of login user as "guest"	binary	1
25	count_same_dest	Number of connections to the same destination host in the past 2 seconds	N	511
26	count_same_serv	Number of connections to the same service (port) in the past 2 seconds	N	511
27	perc_dest_error	Proportion of connections of "count_same_dest" that activated "status_flag" as "error" or "reject"	N	1
28	perc_serv_error	Proportion of connections of "count_same_serv" that activated "status_flag" as "error" or "reject"	N	1
29	perc_dest_rej	Proportion of connections of "count_same_dest" that activated "status_flag" as "reject"	N	1
30	perc_serv_rej	Proportion of connections of "count_same_serv" that activated "status_flag" as "reject"	N	1
31	perc_dest_serv	Proportion of connections in common of "count_same_dest" and "count_same_serv"	N	1
32	perc_dest_nserv	Proportion of connections of "count_same_dest" but not aggregated in "count_same_serv"	N	1

continued on next page

Table 4.1 *continued from previous page*

#	Name	Description	Type	Maximum
33	perc_serv_ndest	Proportion of connections of "count_same_serv" but not aggregated in "count_same_dest"	N	1
34	number_same_IP	Number of connections to the same destination host IP address	N	255
35	number_same_hserv	Number of connections to the same destination host service	N	255
36	perc_IP_ same_hserv	Proportion of connections to the same IP ("number_same_IP") that were to the same port	N	1
37	perc_IP_ nsame_hserv	Proportion of connections to the same IP ("number_same_IP") that were not to the same service	N	1
38	perc_hserv_ same_sserv	Proportion of connections to the same host service ("number_same_serv") that were from the same source port	N	1
39	perc_hserv_ same_IP	Proportion of connections to the same host service ("number_same_serv") that were to the same host IP	N	1
40	perc_IP_ error	Proportion of connections to the same IP ("number_same_IP") that activated "status_flag" as "error" or "reject"	N	1
41	perc_hserv_ error	Proportion of connections to the same host service ("number_same_serv") that activated "status_flag" as "error" or "reject"	N	1

continued on next page

Table 4.1 *continued from previous page*

#	Name	Description	Type	Maximum
42	perc_IP_ rej	Proportion of connections to the same IP ("number_same_IP") that activated "status_flag" as"reject"	N	1
43	perc_hserv_ rej	Proportion of connections to the same host service ("number_same_serv") that activated "status_flag" as "reject"	N	1

TABLE 4.1: List of the different features of the NSL-KDD data-set after changing the type. The type column corresponds to the original type, hence the presence of categorical (C). Numerical values are represented by a N and binary by a B.

4.2.3 Training, validation and test sets

Algorithms are always validated using cross-validation[1] and the testing is also done using cross-validation to limit the variance of the performance results: multiple experiments are performed and the mean performance is then taken.

The set is divided into a training set and a test set randomly in a way that they have no common samples. The training sets are of variable sizes (depending on the experiments, cf. infra) and the test sets are always fixed to 10,000 samples. Both the test set and the training set are constructed in the following way: we try to have an equal proportion of the different training classes. When a class has not enough samples to attain this proportion (e.g. U2R or sometimes R2L), the maximum possible number of samples is picked while respecting the fact testing and training set have no common elements. This is done to limit the problems of unequal distributions described here above. However, this cannot resolve the issue perfectly as some classes U2R have not enough samples to be comprised in the same proportions as the others.

When validation is needed, the training set is again randomly divided in a new smaller training and validation set (without taking care of proportions). The test sets are never touched to construct the validation sets.

[1]Instead of using one validation set, k-fold cross-validation uses k different training and validation sets sampled from the original training set and performs k different experiments. This is done to try to reduce the dependence on the specific sampling of one training and validation set. 1-fold cross-validation corresponds to normal validation with a validation set and n-fold where n is the size of the data-set is called *leave-one-out*. The more folds are used, the less variant are the performance results, but the more it costs to execute the experiments as they are executed k times.

4.3 Support Vector Machines

Let us first benchmark the support vector machines and see what improvements (reductions) can be made to increase their evaluation time in an MPC setting.

4.3.1 Linear support vector machines

Figure 4.1 shows the results of a single support vector machine with a linear kernel function. A first observation is that the tree-based classifier does not perform worse that the one-against-all model. Indeed, the order of the SVMs in the tree-based model is very important as it can make it perform worse or better than one-against-all model. In this sense, the best tree-based structure does perform better and is supposed to have a faster evaluation using MPC as it comprises one SVM less.

Secondly, as we can see, the results are quite satisfying. The accuracy is high, nevertheless, the results must be nuanced. Indeed, as discussed before, the accuracy doesn't take into account the initial distribution of the classes. This is very important if the initial distribution the classes is not uniform, as in our case. We therefore must have a more detailed look at the results (table 4.2, more results for other training set sizes are given in appendix B.1.1).

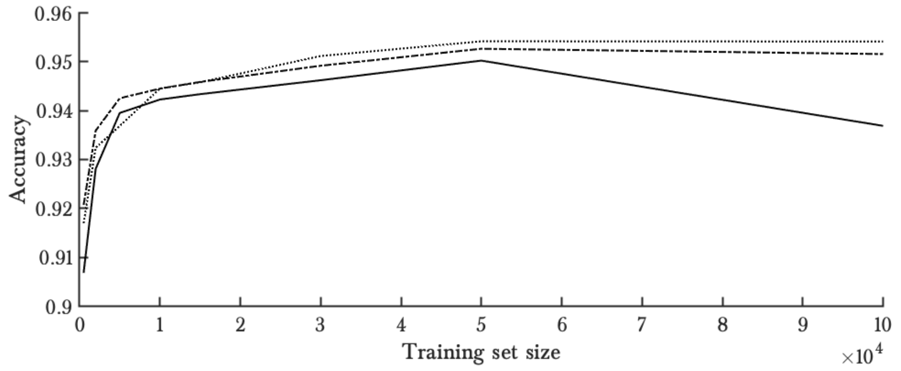


FIGURE 4.1: Evaluation of three different models in function of the training set size. The one-against-all model is in dash-dotted line, the tree-bases model are the plain and dotted line. For the plain line, the order of the SVMs is {Normal, DoS, Prob, R2L, U2R} and the dotted line is {Probe, U2R, R2L, DoS, Normal}. Every result is the mean of 5 different experiments with different training and test set.

A first observation is that the very low number of U2R and in a lesser extent of R2L instances in the training set size lead to a not very satisfying classification. The data of this class is not able to be classified well, this leads to a low MCC coefficient. However, the wrong results are mainly attributed to the normal class. This makes sense as the low number of instances makes the classifier not really able to generalize the properties of this class, not being able to recognize it well and thus unable to distinguish from a normal instance. However, the fact that the false negatives being mainly attributed to the normal class and not randomly assigned to the other classes explains the high kappa coefficient. This example justifies the use of those two coefficients together as

Model	Normal	Probe	DoS	R2L	U2R	Total
Tree 1 with $n = 30,000$						
Accuracy [%]	91.37	97.40	96.34	94.07	73.33	94.62
MCC [%]	89.00	96.03	93.15	85.20	67.38	86.15
Kappa [%]	27.06	42.34	42.25	93.59	99.66	83.19
Obs. Normal	2741	63	136	55	5	
Obs. Probe	53	2195	2	3	0	
Obs. DoS	76	5	2171	1	0	
Obs. R2L	13	0	0	213	0	
Obs. U2R	2	0	0	1	9	
Tree 2 with $n = 30,000$						
Accuracy [%]	92.56	97.11	96.18	93.27	66.67	95.12
MCC [%]	89.84	96.82	92.39	87.16	73.46	89.12
Kappa [%]	26.35	42.70	42.14	93.79	99.72	84.74
Obs. Normal	2777	29	151	42	2	
Obs. Probe	56	2189	9	0	0	
Obs. DoS	80	6	2168	0	0	
Obs. R2L	14	1	0	211	0	
Obs. U2R	0	0	0	4	8	
O-A-A with $n = 30,000$						
Accuracy [%]	93.03	96.67	96.53	94.60	70	94.62
MCC [%]	90.13	96.84	92.81	88.31	77.52	87.93
Kappa [%]	26.07	42.96	42.05	93.78	99.72	84.12
Obs. Normal	2791	18	150	40	1	
Obs. Probe	71	2179	4	0	0	
Obs. DoS	70	8	2176	0	0	
Obs. R2L	12	0	0	214	0	
Obs. U2R	0	0	0	3	8	

TABLE 4.2: Detailed results of the linear SVM classification algorithm for different multi-class models. The first one is a tree-based model of order {Normal, DoS, Prob, R2L, U2R}, the second of order {Probe, U2R, R2L, DoS, Normal} and the last one a one-against-all model. Every result if the mean of 5 independent experiments.

they are complementary in this example. Not much can be done to solve this issue except massively augmenting the presence of the U2R — and in a lesser way R2L — presence in the training data-set. This low result is also to be nuanced as their scarce appearance is also an indication for their low frequency in real-life cases. We can thus conclude that the model doesn't detect much of these attacks, but hopefully they are scarce.

A second observation is the very low kappa coefficient for the normal class. Checking the details indicates that the normal instance identified as attacks are proportionally distributed among the other classes, which translates into a much higher Matthews correlation coefficient (MCC). In other words, misclassified normal classes don't tend to be identified as one class specifically above another.

A third observation is that the results get better with the training size, which is an expected result in machine learning. However, for small training sizes the one-against-all model performs better than both tree-based models. This is to be nuanced due to the small training size which can lead to a much higher variance in the results.

Overall, there are two key observations The fist one is that one-against-all models are not better than tree-based models, it just depends on the order of the tree. The second one is that training sets larger than 30,000 don't make a lot of difference anymore. These facts certainly matters as the number of support vectors is suspected to increase with the training set size, but it is to be nuanced as linear SVMs only depend on the feature size in their primal form and not on the number of support vectors. However, this will have an impact when the evaluation is done in the dual, e.g. RBFSVMs — but this is a talk for section 4.3.2.

PCA reduction

Let us now investigate how a principal components decomposition affects the accuracy and allows us to win execution time. The variance contribution of the first principal components is given in figure 4.2. As the variance contribution is not drastically decreasing, this plots indicates that most of the features are relevant and not so much of them are due to linear combinations of the others features. This also suggests by consequence that we will not be able to limit ourselves to a projection into a space of very small dimension. The influence of a varying number of principal components retained — which corresponds to the dimension of the projected space — is given in figure 4.3. We thereby conclude that we cannot limit ourselves to 6 features as a elbow rule[2] would suggest, but that we need more of them, e.g. 16. The more detailed results for 16 components retained are given in table 4.3 (more details to be found in appendix B.1.2). Here again, we can conclude that there is no significant difference between the tree-based model and the one-against-all model.

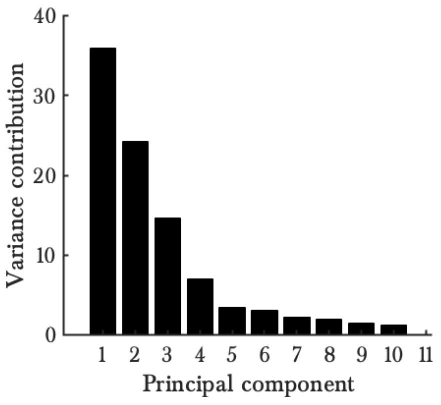


FIGURE 4.2: Variance participation of the 11 first components.

Now, could we only perform one of the operations in a secure manner (PCA or SVM) and do the other one in clear? Let's investigate the consequences of doing this:

- **Secure PCA - Clear SVM**. The first idea of doing the PCA reduction using MPC and evaluating the SVM in clear is not very good. It is very naive to think that PCA destructs the data in such a way that the transformed feature lose most of their information. By transforming using a principal component analysis, we

[2]The number of principal compenents retained after which the variance gain becomes marginal.

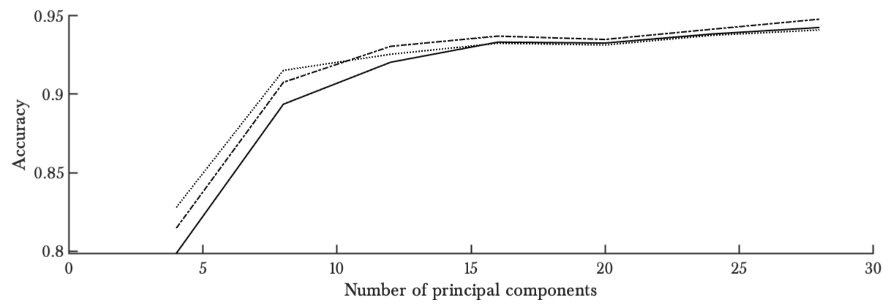


FIGURE 4.3: Evaluation of three different models in function of number of principal components retained. The one-against-all (or parallel) model is in dash-dotted line, the tree-bases model (or parallel) are the plain and dotted line. For the plain line, the order of the SVMs is {Normal, DoS, Prob, R2L, U2R} and the dotted line is {Probe, U2R, R2L, DoS, Normal}. Every result is the mean of 5 different experiments with different training and test sets.

Model	Normal	Probe	DoS	R2L	U2R	Total
Tree with $n_{pca} = 16$						
Accuracy [%]	91.33	94.94	94.19	96.11	33.33	93.24
MCC [%]	87.13	95.35	90.77	79.68	∅	∅
Kappa [%]	26.76	43.51	42.83	93.25	99.62	78.86
Obs. Normal	2740	12	150	89	8	
Obs. Probe	98	2142	16	1	0	
Obs. DoS	100	22	2125	8	1	
Obs. R2L	8	0	0	208	0	
Obs. U2R	6	0	0	4	5	
O-A-A with $n_{pca} = 16$						
Accuracy [%]	91.11	95.55	95.63	95.09	18.67	93.31
MCC [%]	88.16	95.62	91.18	80.59	25.99	76.31
Kappa [%]	27.08	43.23	42.09	93.42	99.74	80.29
Obs. Normal	2733	24	162	80	1	
Obs. Probe	72	2156	26	3	0	
Obs. DoS	80	15	2157	2	1	
Obs. R2L	10	0	1	205	0	
Obs. U2R	6	0	0	6	3	

TABLE 4.3: Detailed results of the linear SVM classification algorithm for different multi-class models with PCA decomposition and $n_{pca} = 16$ components retained. The first one is a tree-based model of order {Normal, DoS, Prob, R2L, U2R} and the second one a one-against-all model. Every result if the mean of 5 independent experiments.

aim at the exact opposite which is to keep as much variance possible, thus as much information as possible. By looking here above at the results of the classifier after the PCA reduction, we notice that the PCA reduction does approach the same classification performance as without. Using clear SVM has also the results that everyone that participates in the evaluation of the transformed data-point can see its classification result.

The clear SVM evaluation should therefore be limited to the strict minimum parties, i.e. the model owner sending the SVM-model to the query owner which then performs the classification on its own. In this way, his data-point is always hidden, but the model owner reveals a part of his model. However, this model part is unusable in itself as it needs the PCA reduction, which is still hidden.

The other possibility to keep the entire model (PCA and SVM) hidden is that the query owner sends its query after PCA to the model owner for him to perform the classification. The owner of the model data, once he knows the transformation of the query point, cannot recover it completely. One can notice than computing the inverse of the PCA reduction will lead to some noisy pre-image in the sense that the pre-image is a not a single data-point but a distribution with variance corresponding to the variance list with the PCA reduction. To summarize, this will of course reveal the final output (the eventual attack and its type), but somewhat hide the features about the query (e.g. duration or status of the connection). This justifies the preference for the model owner sending the SVM-model to the query owner which then performs the classification on its own. In this scenario, the owner however keeps all its model protected.

Finally, one should add that this whole idea of computing the principal component analysis in MPC and then the SVM in clear is computationally doubtful: the MPC-based PCA-transformation of the query point demands $\mathcal{O}\left(n_{pca}d\right)$ MPC multiplication operations where n_f is the feature size and n_{pca} the number of principal components retained, here more than 6 for reasonable performance. Evaluating multi-class support vector machines without PCA and totally hidden has a complexity of $\mathcal{O}(n_{svm}d)$ MPC multiplication operations where n_{svm} is the number of SVMs in the multi-class model, here 4 or 5 depending on which model structure is used. Once the PCA reduction is complete, you still have to add the designation of a winner between the different binary SVMs output which takes $\mathcal{O}(n_{svm})$ MPC comparison operations which are known to be more expensive than multiplications. The question is if the fact that $n_{pca} > n_{svm}$ is compensated by the hidden election of a winner.

- **Clear PCA - Secure SVM**. This alternative has the same property as before, the model is still partially hidden. Furthermore, the hidden evaluation of the SVM guarantees the privacy of the final classification output. In the contrary to here above, the question of who evaluates the PCA decomposition has only one possibility, the user performing the query: the opposite would reveal the initial feature vector to someone else and loose all the privacy of the queried point.

  The first question if we reveal the PCA components is what information is revealed. Well, not much as this is the result of the only diagonalization of the correlation matrix of the model owner's initial data-points. The correlation matrix intrinsically reveals the linear relations between different features. This would barely reveal anything about the original data-points as this relation is computed among ideally several thousands of points from all classes all together.

Furthermore, this correlation matrix is diagonalized and the only first components are retained, revealing in a certain way the only important relations. With the hypothesis that each class has different subjacent relations and thus different principal components, an attacker could — but this is going quite far — deduce which classes were present in the model owner original data-set and thus deduce to which attack he is subject to, or at least detected. However, the exact nature of the original individual data-points remains secret and so does the queried data-point as it never leaves its owner in clear form.

The advantage of this method is that performing an hidden SVM with pre-computed PCA reduction reduces the complexity from $\mathcal{O}(n_{svm}n_f)$ to $\mathcal{O}(n_{svm}n_{pca})$ and in the practice of a factor between 2 and 4. However, the designation of the winning SVM remains $\mathcal{O}(n_{svm})$.

- **No PCA - Clear SVM.** In this case, the query owner receives the entire model from the model owner. Of course, not much here is still hidden. In fact, all the model is clear an can be used by anybody who receives it and there is no need at all for MPC techniques. However linear SVM have the big advantage against the other ones, using kernel trick, that it doesn't need the dual to be computed. This means that instead of sending the α_i with their corresponding support vectors x_i and targets l_i (which will be of course a huge breach of privacy-friendliness), one can just send the weights vector w (and the bias b), which reveals much less information about the model. The exact relation between the weights and the support vectors is

$$w = \sum_{\forall i} \alpha_i y_i x_i. \tag{4.7}$$

The bias stays the same. The computation of the weights is of course a very big compression of the support vectors and destroys the information they contain. In this sense, a totally clear model could be an option, depending on the choices of the model owner. This has no competitor considering the execution speed among the models we test. Indeed, avoiding the use of MPC leads to drastic speed improvements. There is however a drawback less related to privacy-friendliness but is still relevant: the model owner loses control over his model. Indeed, the big advantage of using MPC on a significant part of the model is that the other celar parts of the model are unusable without the MPC-part, whose parameters are never revealed. In this sense, the owner of the data still has a full control as he is the only one who can ultimately decide whether or not his model can be used. Once all is in clear, this property disappears and there is no more secure lock against the proliferation of the models use without its consent. This can be relevant for commercial uses for example[3].

[3]Besides, this raises an interesting question on how to secure software against illegal proliferation and use. Instead of using license keys, why just not performing a very small, but essential task in MPC without which the program could possibly work (in an information-theoretic scheme). In this sense, the owner is sure that only identified users — which he can control at each moment — are allowed to use the

χ^2-reduction

Where the PCA decomposition is the same for all binary SVM, the χ^2 feature selection allows us to select the most relevant features for each SVM. This allows more specifically pre-processed classifiers. Here, the big advantage is a reduction of complexity from $\mathcal{O}(n_{svm}n_f)$ to $\mathcal{O}(n_{svm}n_{\chi^2})$ where n_{χ^2} s the number of features retained. In our case, the reduction results in a reduction of 30% of the feature inputs and thus an expected similar speed gain. Table 4.4 shows the execution of the three models and we can see that there is no significant loss of accuracy compared to the other models. However, the training of a χ^2 selection is much slower because it requires to train n_f support vector machines per binary classification instead of one. However, as we said, the evaluation time got a speed increase. In other words, we won at evaluating time at the cost of training time. The good part is that the training is noramlly only done once, conversely to the MPC-evaluation which is proportionally much more costly. This is a clear example of the training-evaluating trade-off we try to take advantage of.

All these experiments also allow us to gain some insight on the relevance of each input feature. Figure 4.4 shows the relevance of each input feature based on the different methods we tested. We can for example see that some features are very relevant while others not at all. For example, the proportion of connections to the same IP address or the type of protocol used are very relevant for the classification while the number of shell prompts or the consistency between IP addresses seem not very relevant.

A last thing, performing a k-means clustering algorithm isn't very interesting in the case of support vector machines as the sole parameter influencing the evaluation time are the support vectors close to the decision boundary, by essence of support vector machines. The number of data-points far from the decision boundary — which we try to reduce with a k-means algorithm — has thus no influence on the number of support vectors. The only eventual gain of reducing the data-set size with a k-means algorithm is to reduce training time, but this is not much of our concern here.

Secure evaluation

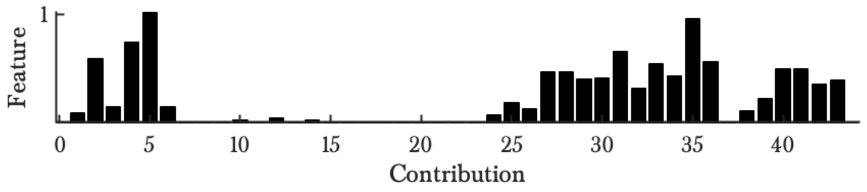
We can now estimate the different models according to the 4 performance indicators described in section 3.2.6 (figure 4.5). As observed in the previous section, $n = 30,000$ samples are a good value for the training set size as the learning performance doesn't increase significantly for higher values. All the models tested are thus based on this value. The models tested are

program. This of course would need to rest on a permanent internet connection (which is nowadays less an issue, more and more smartphone apps are requiring a permanent internet connection) and avoiding the MPC task being cracked and substituted by local version. This is a totally out of the scope of my thesis, but I believe the question to be interesting. Of course, this idea could also be used without MPC and just data sent in an encrypted form, but this would not be privacy-friendly for the user. Another question would also be: who should the eventual third party be to avoid any malicious majority (the first party being the user and the second the company). The third party would be a party that has no interest in working with the user and thus cracking the program nor with the firm issuing the program revealing the user's data. This could be replaced by homomorphic encryption to avoid the use a of third party but is computationally more costly.

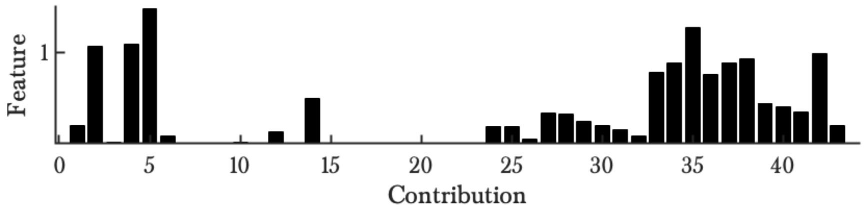
Model	Normal	Probe	DoS	R2L	U2R	Total
Tree with χ^2 and $n = 30,000$						
Accuracy [%]	91.57	97.83	96.24	92.89	87.50	94.79
MCC [%]	89.41	97.00	92.74	83.52	61.82	84.90
Kappa [%]	26.93	42.13	42.00	93.97	99.69	83.73
Obs. Normal	2747	37	146	61	9	
Obs. Probe	45	2211	3	1	0	
Obs. DoS	75	10	2175	0	0	
Obs. R2L	14	0	1	196	0	
Obs. U2R	1	0	0	0	7	
O-A-A with χ^2 and $n = 30,000$						
Accuracy [%]	92.20	97.17	96.55	91.47	50	94.89
MCC [%]	89.93	97.12	92.15	83.73	63.22	85.23
Kappa [%]	26.57	42.54	41.67	94.08	99.83	83.93
Obs. Normal	2766	24	155	54	1	
Obs. Probe	43	2196	20	1	0	
Obs. DoS	74	4	2182	0	0	
Obs. R2L	17	0	1	193	0	
Obs. U2R	1	0	2	1	4	

TABLE 4.4: Detailed results of the linear SVM classification algorithm for different multi-class models with χ^2 feature selection. The first one is a tree-based model of order {Normal, DoS, Prob, R2L, U2R} and the second one a one-against-all model. Every result if the mean of 5 independent experiments.

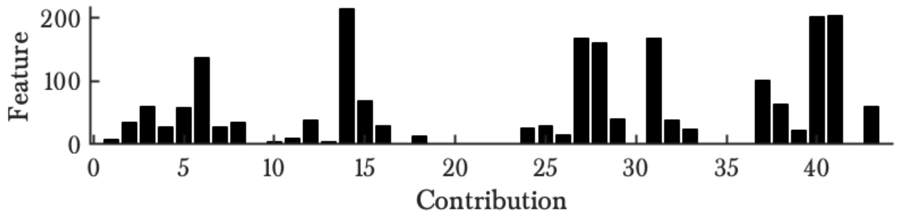
- **(P)LSVM**: secure linear support vector machine;
- **(P)PCA8-(P)LSVM**: secure linear support vector machine with secure principal component analysis decomposition with 8 principal components retained;
- **(P)PCA16-(P)LSVM**: secure linear support vector machine with secure principal component analysis decomposition with 16 principal components retained;
- **(P)PCA8-(C)LSVM**: clear linear support vector machine with secure principal component analysis decomposition with 8 principal components retained;
- **(P)PCA16-(C)LSVM**: clear linear support vector machine with secure principal component analysis decomposition with 16 principal components retained;
- **(P)PCA8-(C)LSVM**: secure linear support vector machine with clear principal component analysis decomposition with 8 principal components retained;
- **(P)PCA16-(C)LSVM**: secure linear support vector machine with clear principal component analysis decomposition with 16 principal components retained;
- χ^2-**(P)LSVM**: secure linear support vector machine with χ^2 feature selection.



(A) Mean of the 6 first PCA coefficients.



(B) Mean of the first 16 PCA coefficients.



(C) Mean of the χ^2 measure on all the model's SVMs.

FIGURE 4.4: Feature relevance using the χ^2-measure on linear SVMs.

As predicted we can observe that performing a secure PCA decomposition together with a secure support vector machine is not very interesting. The accuracy is worse and we increase all cost indicators.

Similarly, performing a secure PCA decomposition using a clear SVM is not interesting considering the accuracy and the computational cost. The big advantage is that the clear computation of support vector machines avoids the need for secure comparisons (when evaluating the multi-class model) and drastically reduces the number of rounds and communication cost. This could maybe be interesting for very high-latency or slow networks, though the advantage seems relatively limited considering the costs including the revelation of the SVM model parameters.

As predicted again, the clear PCA decomposition together with secure SVM evaluation seems an interesting alternative. The need for comparisons still results in a high round cost, however the lower dimension of the feature vector drastically reduces the computational and the communication cost. However, the accuracy is still lower than the full model and the PCA principal components may reveal (limited) information about the training set. This model could be considered if absolute secrecy about the training isn't required as some statistical information about it could be deduced from the PCA coefficients. For the $n_{pca} = 16$, the accuracy is not as good as the full model, but not significantly lower.

The best model here seems to be the χ^2 feature selection with full secrecy. The

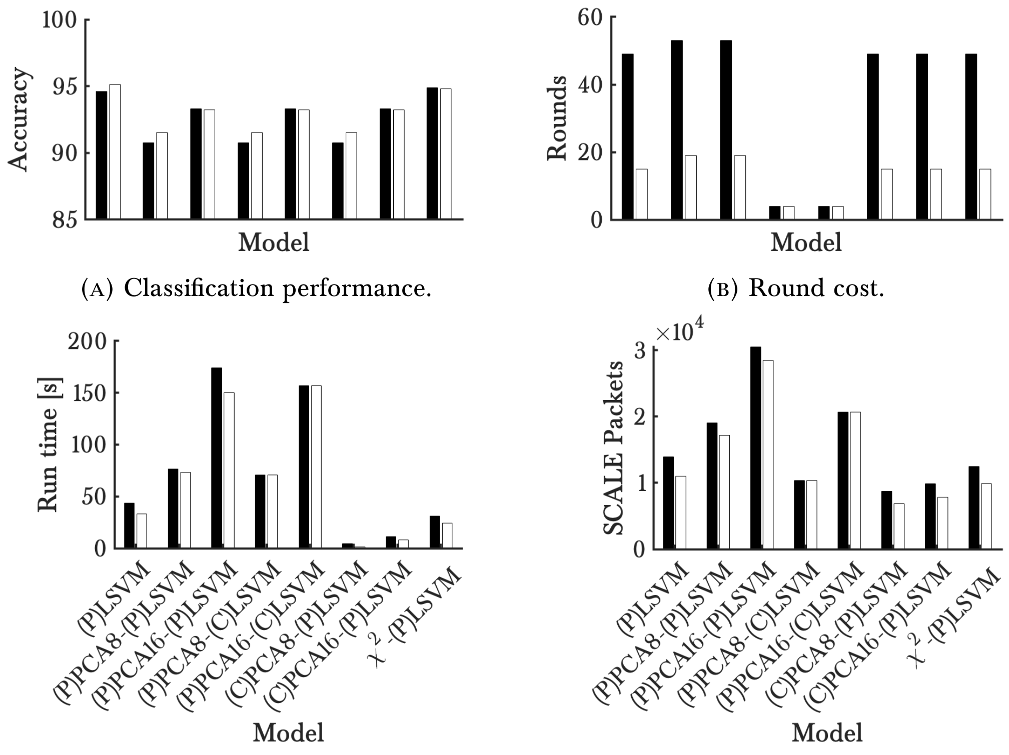


FIGURE 4.5: Comparison of different protocols for secure linear support vector machine evaluation with and without various feature size reduction methods. The black bars correspond to the one-against-all multi-class model and the white ones to the tree-based multi-class model. The results correspond here of the secret evaluation of 10 queries.

accuracy is as good as the full model if not better and there is a significant reduction of the computational and communication cost. The round cost is similar as the other models due to the need for comparisons.

Furthermore and as predicted, the tree-based model seems to perform in general as well as the one-against-all model, but allows for improvements in all other indicators, the most significant being the round cost.

To summarize the case of secure linear support vector machines for intrusion detection systems, the model to be preferred is the tree-based model with χ^2 feature selection.

4.3.2 Non-linear support vector machines

Now that we have benchmarked the linear model, we can have a look at some non-linear models. Figure 4.6 shows the results of a radial based function support vector machine on the NSL-KDD data-set. The box constraints C and the kernel function parameter σ^2 are optimized at each specific training through a 10-fold cross-validation.

A first observation compared to the linear support vector machines models is the much better accuracy. This is due to the added non-linearity which allows more complexity. The individual scores of the support vector machines are attaining values very close to 100%. We can observe that accuracy and Cohen's kappa coefficient are following almost exactly the same graph, at the difference of a factor. This is typical when the models are attaining very high accuracies and make little mistakes. This comforts us in our claim of a good model.

A second observation is the same we made here before with the linear support vector machines: the tree-based model are performing worse than the one-against-all model. Here again, the accuracy doesn't increase much more after $n = 15,000$. A difference with before is the better performance of the other sequence of the tree-based model, even though the difference is minimal. Results for the best tree-based model and the one-against-all model are given in table 4.5 (more in appendix B.1.3).

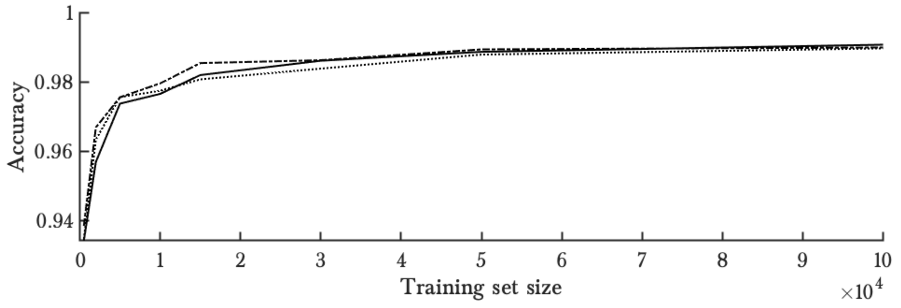


FIGURE 4.6: Evaluation of three different models in function of the training set size. The one-against-all (or parallel) model is in dash-dotted line, the tree-bases model (or parallel) are the plain and dotted line. For the plain line, the order of the SVMs is {Normal, DoS, Prob, R2L, U2R} and the dotted line is {Probe, U2R, R2L, DoS, Normal}. Every result is the mean of 5 different experiments with different training and test set.

Support vector reduction

A difference with linear support vector machines is that the use of a kernel matrix doesn't allow us to evaluate a new data-point in the primal space anymore. This primal space had the dimension of the feature space and the training size had not a lot of influence on the evaluation of a new query point as a consequence. However, this is not the case anymore and the evaluation now happens in a support vector space. As a reminder, the evaluation of a new data-point x in the SVM is now given by

$$\sum_{i=1}^{n_{sv}} \alpha_i y_i K(x, x_i), \tag{4.8}$$

where $K(x, x_i)$ is the kernel function and n_{sv} the number of support vectors. For each of our models trained before, the number of support vectors is given in figure 4.7. The number of support vectors is much higher for the one-against-all model as suspected as it has one more support vector machine than the tree-based model. However, an interesting observation is that both tree-based models — although having the same

Model	Normal	Probe	DoS	R2L	U2R	Total
Tree 1 with $n = 15,000$						
Accuracy [%]	97.82	99.08	99.01	91.71	24.00	98.20
MCC [%]	96.62	99.03	98.54	87.47	32.25	82.78
Kappa [%]	23.65	42.23	42.14	93.71	99.69	94.39
Obs. Normal	2935	7	21	33	5	
Obs. Probe	18	2229	2	0	0	
Obs. DoS	19	2	2228	1	0	
Obs. R2L	16	1	1	215	1	
Obs. U2R	5	0	0	6	4	
Tree 2 with $n = 15,000$						
Accuracy [%]	97.98	98.25	99.20	91.45	28.00	98.08
MCC [%]	96.23	98.44	98.56	89.30	$\emptyset$	$\emptyset$
Kappa [%]	23.46	42.56	42.04	93.85	99.72	94.00
Obs. Normal	2939	8	24	26	2	
Obs. Probe	36	2211	4	0	0	
Obs. DoS	17	1	2232	0	0	
Obs. R2L	19	1	0	214	0	
Obs. U2R	7	0	0	3	4	
O-A-A with $n = 15,000$						
Accuracy [%]	98.33	99.40	99.15	92.14	24.00	98.55
MCC [%]	97.28	99.41	98.84	88.54	33.55	83.52
Kappa [%]	23.38	42.13	42.14	93.76	99.72	95.47
Obs. Normal	2950	4	15	30	1	
Obs. Probe	12	2237	1	0	0	
Obs. DoS	18	1	2231	0	0	
Obs. R2L	15	1	0	216	2	
Obs. U2R	5	0	1	5	4	

TABLE 4.5: Detailed results of the RBF-SVM classification algorithm for different multi-class models for $n = 15,000$. The first one is a tree-based model of order {Normal, DoS, Prob, R2L, U2R} and the second one a one-against-all model. Every result if the mean of 5 independent experiments.

number of support vector machines — have a significant divergence in the total number of support vectors. Furthermore, it is the best of both models that comprises the less number of support vectors. In this latter case, the total number of support vectors never exceeds significantly a thousand. Let's however see if we can reduce this number further.

As said before, C is optimized through validation. This directly controls the number of support vectors. To improve the speed of the evaluation, one must thus reduce the number of support vectors. This can be controlled by the box constraint C which — as a reminder — represents the trade-off between the objective of a support vector machine and the influence of the slack variables — we could call the corresponding

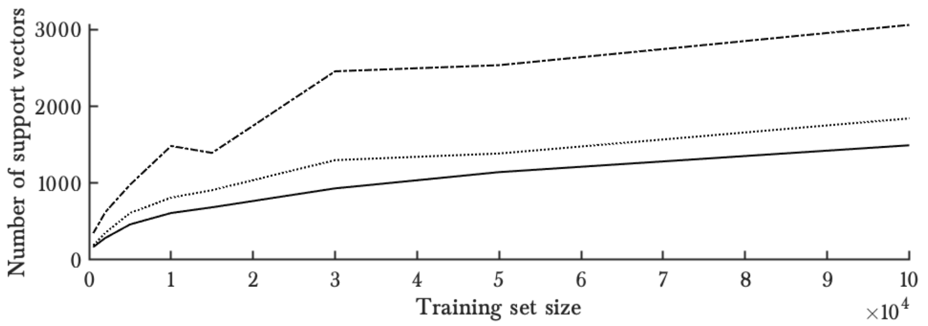
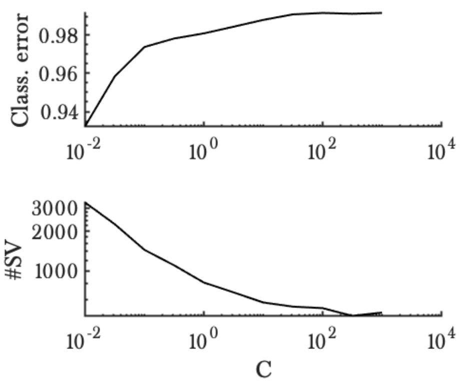


FIGURE 4.7: Total number of support vectors for different models based on the training set size. The one-against-all (or parallel) model is in dash-dotted line, the tree-bases model (or parallel) are the plain and dotted line. For the plain line, the order of the SVMs is {Normal, DoS, Prob, R2L, U2R} and the dotted line is {Probe, U2R, R2L, DoS, Normal}. Every result is the mean of 5 different experiments with different training and test sets.

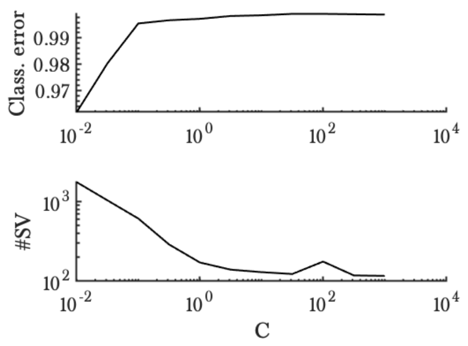
data-points the "difficult" ones. In this sense, a high value of C will just make the boundary absolutely fit to every variable and tolerate no wrongly classified data-point to the model. In other words, the difficult data-points will have a much higher influence. This is even more clear in the dual as the box constraint C directly represents an upper bound on the α_i. A high value of C leads to less regularization and the model to fit to these specific difficult points, which increases the risk of overfitting. Let's investigate how far we can increase the box constraint C without suffering from overfitting. This way, we can reduce the MPC evaluation without having too much impact on the accuracy.

Figure 4.8 shows how the number of support vector decreases as the box constraint parameter C increases. The result of the last SVM shows more variability as the low number of data-points of the classes R2L and U2R which it classifies. In general, we can observe that the after a certain value, the number of support vectors doesn't decrease anymore and the accuracy, which is almost perfect, stagnates. However, we see that we aren't really subject to overfitting. The stagnation of the number of support vectors and the absence of ovefitting tells us that all data-points are within the good side of the boundary. In this sense, imposing the misclassified ones to fit to the boundary has no effect, hence the stagnation. In a certain sense, this corresponds to the ideal boundary. What is more, the very high box constraint C also diminishes the impact of the first term of the boundary, which controls its smoothening. However, finding the optimal C without imposing it to be high already naturally tends to a high value. This indicates that the optimal boundary without the constraint of lowering the number of support vectors is not smooth. In this sense, the boundary is only defined by the critical data-points. These sole critical data-points are the only ones the support vectors consists of. This is also the reason for the stagnation of their number.

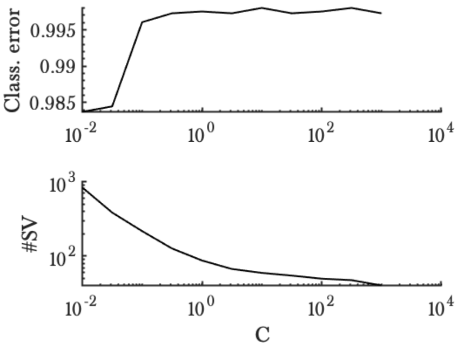
By taking a very high value of C to reduce the number of support vectors, we still obtain an accuracy similar to before (table 4.6 compared to table 4.5). However the number of support vectors went from 931 with the classically validated C to 588 with a high value of C. This is a decrease of 37% and we can expect a similar increase in time. For the other tree-based model, the number of support vectors went from 908 to



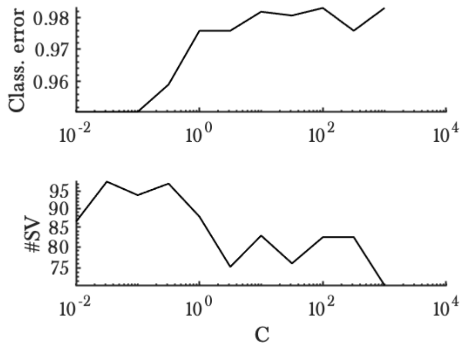
(A) First SVM.



(B) Second SVM.



(C) Third SVM.



(D) Fourth SVM.

FIGURE 4.8: Support vector reduction for the tree-based model with SVM order {Normal, DoS, Prob, R2L, U2R} with $n = 15,000$ data-points.

643 representing a decrease of 29% For the one-against-all model, it went from 1393 to 1031 representing a decrease of 35%.

PCA with support vector reduction

Each tested data-point has to be computed against all support vectors. Now that the number of support vectors has been reduced, let us investigate if we can also reduce the evaluation of each support vector in itself. As a reminder, the radial based kernel function is given by

$$K(x, y) = e^{\frac{\|x-y\|^2}{2\sigma^2}}, \tag{4.9}$$

and the norm is given by

$$\|x - y\| = \sum_{j=1}^{n_f}(x_j - y_j)^2, \tag{4.10}$$

where n_f is the number of features.

Model	Normal	Probe	DoS	R2L	U2R	Total
Tree with $n = 15,000$ and support vector reduction						
Accuracy [%]	98.69	99.71	99.25	90.77	74	98.89
MCC [%]	97.93	99.51	99.00	90.79	63.39	90.13
Kappa [%]	23.16	41.78	41.92	94.32	99.70	96.54
Obs. Normal	2961	7	11	18	4	
Obs. Probe	3	2249	4	0	0	
Obs. DoS	16	1	2239	0	0	
Obs. R2L	17	1	0	201	2	
Obs. U2R	2	0	0	1	7	
O-A-A with $n = 15,000$ and support vector reduction						
Accuracy [%]	98.72	99.69	99.34	92.04	74	98.96
MCC [%]	98.07	99.61	98.98	91.60	63.42	90.34
Kappa [%]	23.16	41.81	41.87	94.29	99.69	96.75
Obs. Normal	2962	4	13	17	4	
Obs. Probe	4	2249	3	0	0	
Obs. DoS	14	1	2241	0	0	
Obs. R2L	14	0	1	203	2	
Obs. U2R	1	0	1	1	7	

Table 4.6: Detailed results of the RBF-SVM classification algorithm for different multi-class models for $n = 15,000$ with support vector reduction. The first one is a tree-based model of order {Normal, DoS, Prob, R2L, U2R} and the second one a one-against-all model. Every result if the mean of 5 independent experiments.

By applying a PCA decomposition, we would be able to approximate the norm and limit the sum to n_{pca} terms instead of n_f terms. The evaluation against each support vector would then be reduced. The cost of doing this is the need to transform each tested data-point through PCA. As we saw before this evaluation is of complexity $\mathcal{O}(n_{pca}n_f)$. Contrary to before, this is far smaller than the complexity needed to evaluate the SVM due to the much higher number of support vectors. The need for evaluating one of the two in clear is thus not justified anymore. Indeed, evaluating the SVM-part in clear would reveal the feature vectors which are highly sensitive data and the evaluation of the PCA-part in clear is significantly less costly than the SVM. We will thus only interest us in evaluating the whole (PCA and SVM) in MPC.

Furthermore, the norm in the feature space gives the same weight to each feature. Using a PCA decomposition will vary the weight as the principal components of the transformed feature are a scalar product of the original feature vector with the PCA coefficients, which can be seen as weights.

Results of the PCA decomposition with RBF-SVM are given in figure 4.9. Compared to the PCA reduction with linear support vector machines, a lower number of principal components retained is feasible here (8 gave not satisfying results with the SVMs with linear kernel, and seems here satisfying). A low number of principal components seems also to impose a higher number of support vectors. This makes sense as the more information loss during the PCA reduction has to be compensated by keeping more information in the SVM under the form of support vectors. Table 4.7 shows the results with 16 principal components. We can clearly see that the performance is as good as without any PCA reduction.

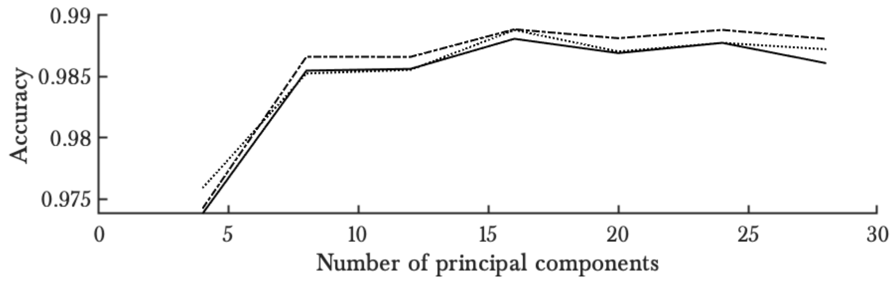


FIGURE 4.9: Influence of the number of principal components retained in a RBF-SVM with support with support vector reduction and PCA reduction. The one-against-all (or parallel) model is in dash-dotted line, the tree-bases model (or parallel) are the plain and dotted line. For the plain line, the order of the SVMs is {Normal, DoS, Prob, R2L, U2R} and the dotted line is {Probe, U2R, R2L, DoS, Normal}. Every result is the mean of 5 different experiments with different training and test set.

χ^2 *with support vector reduction*

Similarly, to limit the number of terms in the sum needed for the norm, we could also perform a χ^2 feature selection as we did before. Computing the selected features with radial based kernel function support vector machines gives very similar results as with the linear one. However, although both methods are keeping 33 of the 43 features, these are not exactly the same. This indicates that some features have a more linear impact (linear kernel functions) on the output and some others have a more local impact (radial based kernels functions). For the rest, apart from a few exceptions, the retained inputs are the same, indicating their clear contribution independently how this contribution is taken into account. Figure 4.10 shows the χ^2 measure of each SVM of a tree-based model. It is very clear that the model with few training data obtains much lower scores.

Another detail is that a feasible cut-off value on the χ^2 measure with RBFSVM is much higher than with linear SVMs. This indicates that the corresponding certainty[4] is much higher. But the most interesting part is that here again, reducing the number

[4]The χ^2 measure used here is based on the classical χ^2 test for categorical data which uses a p-value relevance estimation. Here, we cannot use this test directly as we don't have categorical data as feature inputs. In a certain sense, for each of our tests on each support vector machine, we have 4 input classes used (t_p, t_n, f_p, f_n) and two output classes (1 or 0). This corresponds to a χ^2 with three degrees of freedom. The cut-off value chosen here (10) corresponds to a p-value of .018566. This is statistically satisfying. However, this is not the goal we pursue here. The only thing that matters is finding clever ways to improve the speed of the evaluation of a test-set through our MPC models. For the linear based SVM, the p-value was not satisfying ($p > 0.5$). However, it still allowed us to improve the speed without loosing in accuracy.

Model	Normal	Probe	DoS	R2L	U2R	Total
Tree with $n = 15,000$, $n_{pca} = 16$ and support vector reduction						
Accuracy [%]	98.31	99.55	99.00	97.40	54.29	98.81
MCC [%]	97.74	99.42	98.65	92.45	57.14	89.08
Kappa [%]	23.38	41.75	41.88	94.17	99.83	96.28
Obs. Normal	2949	7	17	23	3	
Obs. Probe	4	2249	3	3	0	
Obs. DoS	20	1	2236	2	0	
Obs. R2L	5	0	0	209	0	
Obs. U2R	3	0	0	1	4	
O-A-A with $n = 15,000$, $n_{pca} = 16$ and support vector reduction						
Accuracy [%]	98.58	99.64	99.01	95.72	45.71	98.89
MCC [%]	97.83	99.56	98.71	93.47	42.85	86.48
Kappa [%]	23.20	41.73	41.89	94.33	99.80	96.52
Obs. Normal	2957	5	16	16	6	
Obs. Probe	4	2251	3	1	0	
Obs. DoS	21	1	2237	1	0	
Obs. R2L	9	0	0	206	0	
Obs. U2R	4	0	0	0	3	

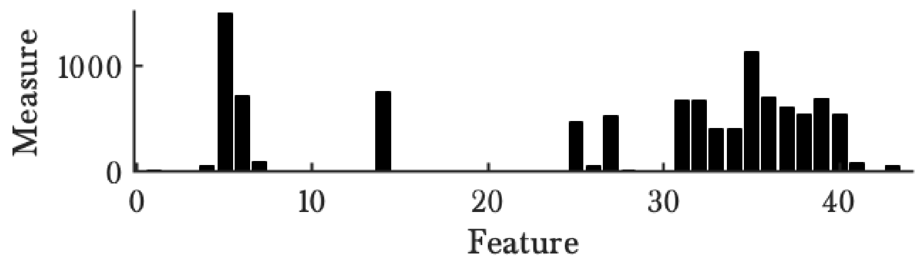
TABLE 4.7: Detailed results of the RBF-SVM classification algorithm for different multi-class models for $n = 15,000$ with support vector reduction and principal component analysis decomposition. The first one is a tree-based model of order {Normal, DoS, Prob, R2L, U2R} and the second one a one-against-all model. Every result if the mean of 5 independent experiments.

of feature inputs by pure selection allows us to keep the same high performance while reducing the model's complexity and evaluation time (table 4.8).

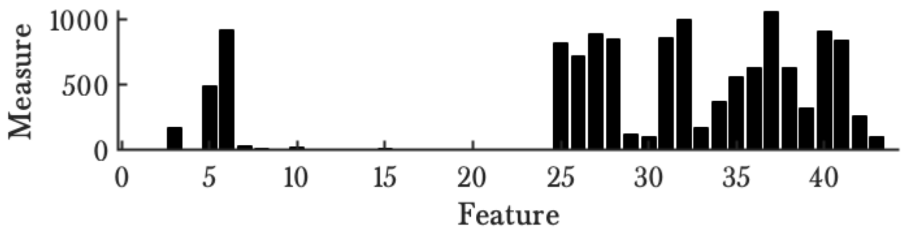
Secure evaluation

Smilarly to linear SVM models, we can now estimate the different models according to the 4 performance indicators (figure 4.11). As observed above, $n = 15,000$ samples are a good value for the training set size as the learning performance doesn't increase significantly for higher values. All the model tested are thus based on this value. The models tested are

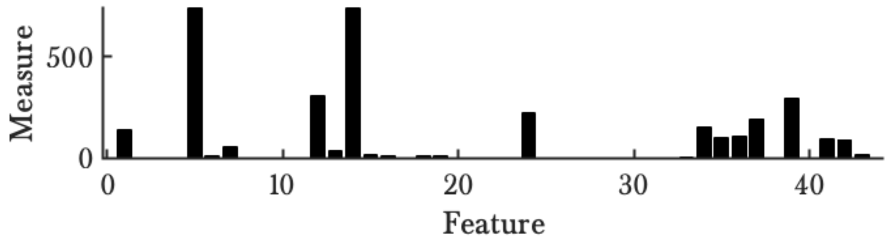
- **(P)RBFSVM**: secure support vector machine with radial based function kernel;
- **(P)PCA8-(P)RBFSVM**: secure support vector machine with radial based function kernel with secure principal component analysis decomposition with 8 principal components retained;
- **(P)PCA16-(P)RBFSVM**: secure support vector machine with radial based function kernel with secure principal component analysis decomposition with 16 principal components retained;



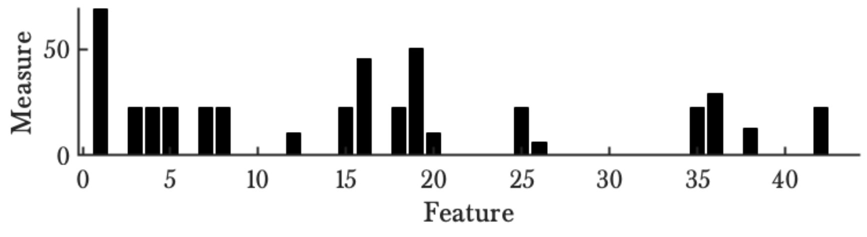
(A) First SVM of the model.



(B) Second SVM of the model.



(C) Third SVM of the model.



(D) Fourth SVM of the model.

FIGURE 4.10: χ^2 measure for each feature in each SVM of the tree-based model with order {Normal, DoS, Prob, R2L, U2R}.

- χ^2-**(P)RBFSVM**: secure support vector machine with radial based function kernel with χ^2 feature selection.

Except for the PCA decomposition with 8 components, all models here have a very good accuracy ($\sim 98\%$) but are demanding much more evaluation time than their counterparts with linear kernel functions. This is caused by the fact briefly evoked here above: the non-linear support vector machines are evaluated in the dual space.

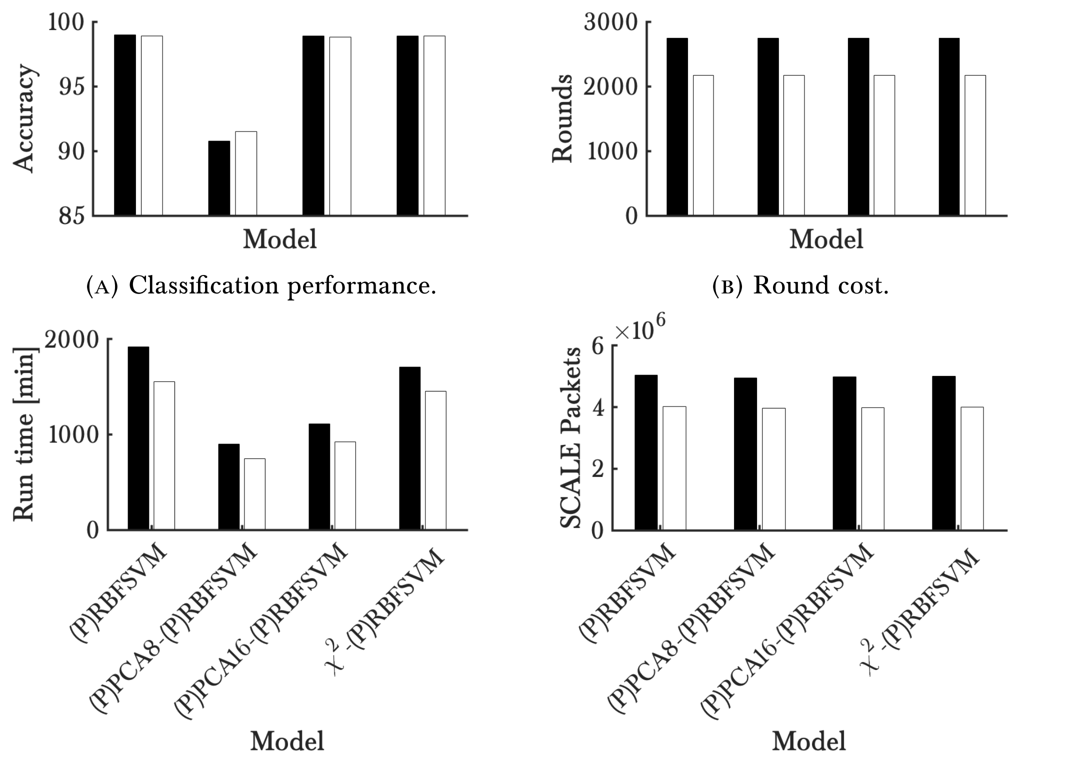
The feature space is not the only speed factor anymore as the evaluation of the

Model	Normal	Probe	DoS	R2L	U2R	Total
Tree with $n = 15,000$, $n_{pca} = 16$ and support vector reduction						
Accuracy [%]	98.55	99.74	99.14	94.14	56.92	98.87
MCC [%]	97.76	99.54	98.75	93.76	60.11	89.98
Kappa [%]	23.25	41.81	41.96	94.26	99.67	96.48
Obs. Normal	2957	6	19	14	5	
Obs. Probe	4	2249	2	0	0	
Obs. DoS	18	1	2236	0	0	
Obs. R2L	12	1	0	209	0	
Obs. U2R	6	0	0	0	7	
O-A-A with $n = 15,000$, $n_{pca} = 16$ and support vector reduction						
Accuracy [%]	98.54	99.73	99.16	94.14	61.54	98.88
MCC [%]	97.81	99.62	98.79	92.57	62.34	90.23
Kappa [%]	23.27	41.83	41.96	94.19	99.67	96.50
Obs. Normal	2956	4	17	19	4	
Obs. Probe	4	2249	1	0	0	
Obs. DoS	17	1	2236	0	0	
Obs. R2L	11	1	2	209	0	
Obs. U2R	5	0	0	0	8	

Table 4.8: Detailed results of the RBF-SVM classification algorithm for different multi-class models for $n = 15,000$ with support vector reduction and χ^2 feature selection. The first one is a tree-based model of order {Normal, DoS, Prob, R2L, U2R} and the second one a one-against-all model. Every result if the mean of 5 independent experiments.

kernel function also has a significant cost, but still plays a role.

The PCA reduction is now marginal to the distance computation with all support vectors and its Gaussian evaluation. This makes the PCA reduction with 16 components the clear winner here though the non-linear SVMs are not very satisfying in general. Similar accuracies can be achieved much faster using the nearest neighbors algorithm investigated here after.



(A) Classification performance. (B) Round cost.

(C) Computational cost. (D) Communication cost.

FIGURE 4.11: Comparison of different protocols for secure support vector machine with radial based kernel function evaluation with and without various feature size reduction methods. The black bars correspond to the one-against-all multi-class model and the white ones to the tree-based multi-class model.

4.4 Nearest neighbours

4.4.1 Classical nearest neighbours

Figure 4.12 shows the results of the k-NN algorithm in function of the number of training data-points for a different number of neighbours. We can see that the performance is very similar to radial based function support vector machines.

A first observation is that contrary to SVMs models, the performance always keeps increasing with added number of training points. This makes sense as support vector machines have a limited complexity due to their limited number of coefficients to be trained in their primal form. This is not totally true for SMVs using kernel functions and being evaluated in the dual, but their complexity is based on the number of support vectors which are consisting of only data-points close to the decision boundary. Adding more training points only improves the performance when those are close to the decision boundary if we neglect the smoothening of the boundary — which in our case appears not to be very important. In k-NN algorithm, all data-points participate to the decision boundary. Increasing the density of data-points allows the decision boundary to be influenced by any new data-point. The need for intricate boundaries — or high complexity — is also an explanation of the good performance of k-NN.

To understand the high variability of Matthew's correlation coefficient, we must take a more detailed look at the individual results (table 4.9, more details in appendix B.2). Matthew's correlation coefficient is computed on the binary correlation matrices for each class. For the total one, the mean is taken. As already mentioned before, the big advantage of this coefficient is that it takes into account the prior probabilities of both binary classes. This has a huge influence on the U2R class — and in a lesser manner the R2L class — that has very few instances. The accuracy is thus very good as very few points are classified in this category as should be for the very big majority, but the classifier is not very successful for the ones that should. As discussed before, not much can be done except increasing the number of instances of these classes.

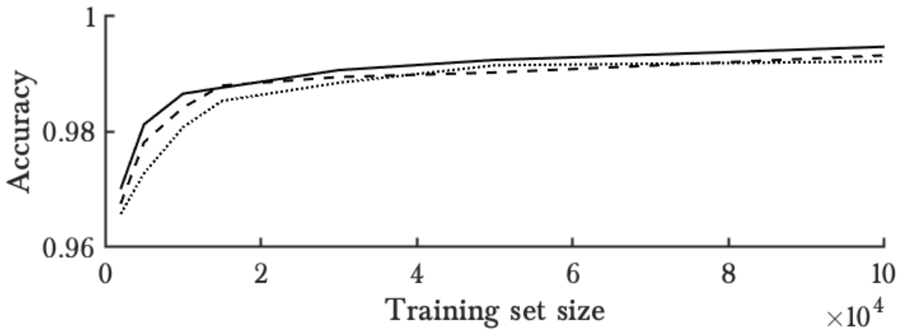


FIGURE 4.12: Performance of the k-NN algorithm in function of the number of training data-points for a different number of neighbours. The plain line is for $k = 1$, the dashed is $k = 2$ and the dotted is $k = 3$.

4.4.2 Training set reduction

The training is linearly dependent on the number of instances in the training set — the ones that are compared to the search for the nearest neighbours. To reduce the evaluation time, we must find a way to reduce this number and the find only the relevant data-points.

k-means clustering

The k-NN does as good as the best radial based function support vector machine for $n = 10,000$ instances and surpasses all the previously tested algorithms for more training points, where this is even more clear for $n = 100,000$.

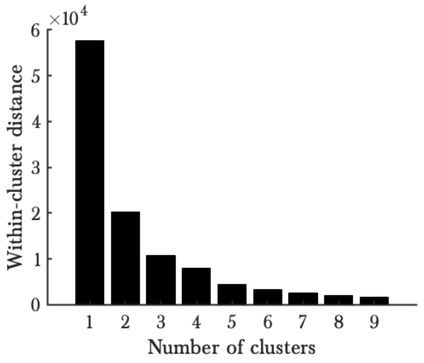


FIGURE 4.13: Within cluster sum of distances in function of the number of clusters chosen.

Model	Normal	Probe	DoS	R2L	U2R	Total
$k = 1$ with $n = 10,000$						
Accuracy [%]	98.17	99.45	98.80	98.03	43.33	98.66
MCC [%]	97.49	99.19	98.63	92.76	39.68	85.55
Kappa [%]	23.48	41.80	42.04	94.07	99.74	95.81
Obs. Normal	2945	11	15	25	4	
Obs. Probe	10	2246	2	1	0	
Obs. DoS	23	3	2231	1	1	
Obs. R2L	2	0	0	214	2	
Obs. U2R	3	0	0	2	4	
$k = 1$ with $n = 100,000$						
Accuracy [%]	99.56	99.82	99.63	93.81	58.33	99.48
MCC [%]	99.00	99.73	99.57	95.39	62.16	91.17
Kappa [%]	22.61	41.51	41.58	94.88	99.85	98.36
Obs. Normal	2987	4	4	5	1	
Obs. Probe	2	2260	2	0	0	
Obs. DoS	8	0	2256	0	0	
Obs. R2L	12	0	0	190	1	
Obs. U2R	2	0	0	1	4	

TABLE 4.9: Detailed results of the k-NN classification algorithm for $k = 1$ for a small and for a large training data-set.

To select the number of clusters, the elbow rule suggests to take 5 clusters, which is by the way the same results obtained by [81], but with a different data-set (figure 4.13). We can then keep only the normal instances near the decision boundary, i.e. the cluster with the closest mean to the other classes. This gives very disappointing results independently of the number of nearest neighbours k chosen. Table 4.10 also shows the results where the sole most-distant cluster is discarded. However, the results are still very disappointing although logically better than keeping the sole best cluster, we can see that a lot of normal data-points are abusively considered as various attacks. The model lacks information about the variety of normal attacks, variety that we just suppressed. This suggests that they are no specific regions where the data-points are not relevant. In other words, the decision boundary goes through all clusters, all regions of the hyper-space where points are present. This once again confirms the complexity of the decision boundary and the problem in general. We could try to increase the number of clusters to discard smaller groups and do this iteratively always discarding the worse one, but after all, the direction this is going to is select the points individually and not based on groups. This problem is better tackled with condensed nearest neighbours.

Condensed nearest neighbours

The clustering-based methods give good results for huge data-sets and has successfully applied it to *random forest classifiers* [81]. In our case, we have a lower number of

Model	Normal	Probe	DoS	R2L	U2R	Total
$k = 1$ with $n = 100,000$ and k-means.						
Accuracy [%]	95.19	99.29	98.78	96.92	76	97.45
MCC [%]	94.88	97.05	98.29	89.79	74.85	90.97
Kappa [%]	25.09	41.02	41.67	94.85	99.60	92.02
Obs. Normal	2856	12	25	5	3	
Obs. Probe	84	2251	2	0	0	
Obs. DoS	23	4	2239	0	0	
Obs. R2L	34	1	0	179	0	
Obs. U2R	4	0	0	0	11	

TABLE 4.10: Detailed results of the k-NN classification algorithm for $k = 1$ and for $n = 100,000$ using k-means data reduction.

data-points and we do not try to discard the majority of them, but intelligently select the relevant ones to reduce our number of instances in the training data-set. The condensed nearest neighbours is an instance-wise selection method and not based on larger groups as are clustering methods. However, the better results of the $k = 1$ does not allow us to discard the outliers.

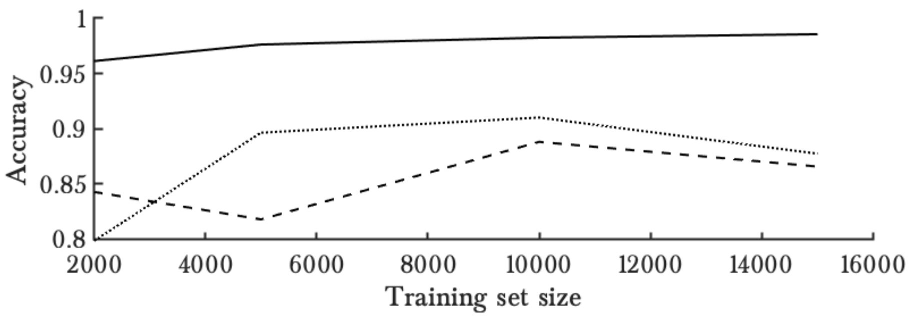
Table 4.11 shows the results of the condensed nearest neighbours which appear to be almost as good as without any data reduction, but a dramatic training size decrease: from 10,000 to less than 400. This represents a data reduction of more than 96% !

Model	Normal	Probe	DoS	R2L	U2R	Total
$k = 1$ with $n = 10,000$ and CNN.						
Accuracy [%]	97.63	99.11	98.26	98.16	80	98.25
MCC [%]	96.73	99.02	98.02	90.27	59.45	88.70
Kappa [%]	23.74	41.94	42.19	93.92	99.70	94.52
Obs. Normal	2929	14	31	3	1	
Obs. Probe	9	2238	1	1	0	
Obs. DoS	20	4	2219	0	0	
Obs. R2L	36	1	4	213	1	
Obs. U2R	6	0	3	0	6	

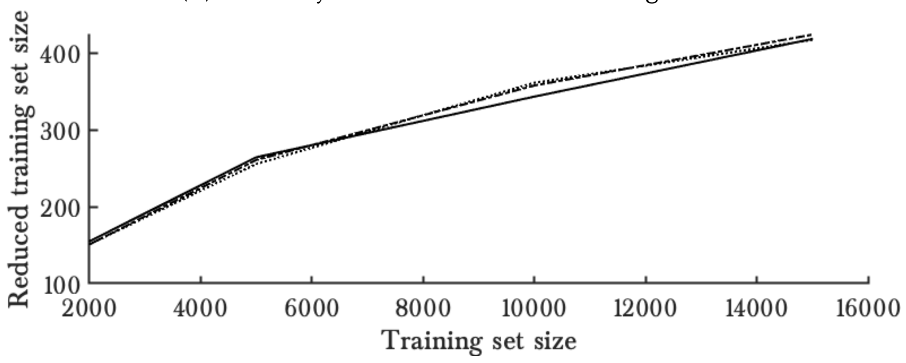
TABLE 4.11: Detailed results of the k-NN classification algorithm for $k = 1$ with condensed neighbours reduction.

4.4.3 Feature size reduction

The k-NN algorithm is composed of two phases: the computation of the distances and the searching for the smallest one. As the second phase will not be impacted by the feature size reduction, this is not the case for the computation of the distances which — as already observed before — increase linearly with the feature size. Of course, the computation of the PCA components has a certain cost, but it becomes marginal in



(A) Accuracy with condensed nearest neighbours.



(B) Number of data-points after reduction.

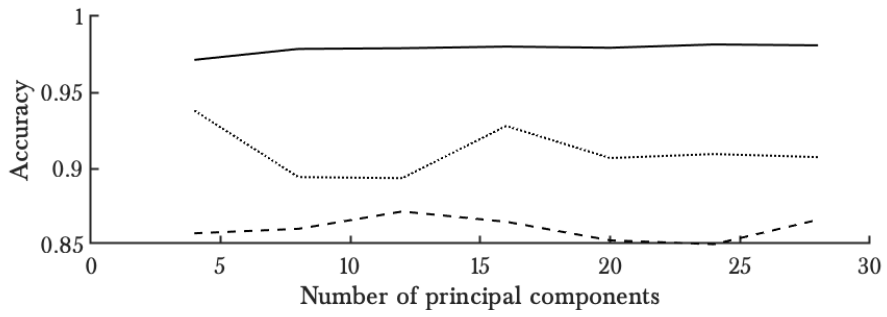
FIGURE 4.14: Performance of the k-NN algorithm in function of the number of training data-points for a different number of neighbours. The plain line is for $k = 1$, the dashed is $k = 2$ and the dotted is $k = 3$.

comparison to the total cost as the number of distances to be computed increases. There is no additional cost for the χ^2 feature reduction but it usually keeps more features.

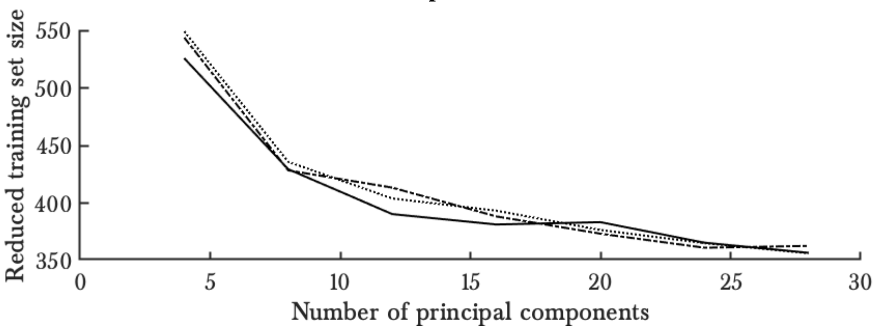
Principal components analysis

In comparison to the support vector machines with nearest neighbors, a much lower number of principal components seems to be needed (figure 4.15). However, the loss of allowed complexity of the model due to the feature size reduction results in more instances to be retained after the CNN reduction. The loss of complexity is compensated by an increase of complexity in the instance dimension, this to allow complex decision boundaries. However, the difference is here relatively marginal: about 430 instances compared to 380. Let us compare the total information retained in the model: $n_t \times (n_p + 1)$ where n_t and n_p are the instance and feature size (number of components) after reduction. The +1 corresponds to class associated with each instance. The case with 8 components corresponds to a total information reduction of almost 47% compared to the case with 16 components. Though, both have pretty much the same classification performance.

However, compared to the CNN without PCA reduction, the accuracy is slightly



(A) Accuracy with condensed nearest neighbours in function of the number of PCA components.



(B) Instance set size after reduction in function of the number of PCA components.

FIGURE 4.15: Performance of the k-NN algorithm with PCA reduction and CNN reduction in function of the number of principal components retained. The plain line is for $k = 1$, the dashed is $k = 2$ and the dotted is $k = 3$.

diminishing, but still resulting in very good results (table 4.12, more results in appendix B.2.3).

χ^2 *feature selection*

Here again, we can also use a χ^2 feature selection instead of a PCA decomposition (figure 4.16 and table 4.13). This gives a better classification performance than the original condensed nearest neighbours (table4.11) for the same size training set size after reduction. The results of the relevance of each feature for each class are given at figure 4.17 and result in similar selections as with SVMs, but the differences are more marked. Here again, we select 33 features based on the scores using the same threshold of 10 on the mean measure of each feature[5]. We have also tested with setting a higher threshold (100) of the χ^2 measure and select only 20 features, but results in lower performance than the PCA reduction with $n_{pca} = 16$ leading to discard this option (appendix B.2.4).

[5]The same 33 features are selected as RBF-SVMs with the same resulting p-value

Model	Normal	Probe	DoS	R2L	U2R	Total
$k = 1$ with $n = 10,000$, $n_{pca} = 8$ and CNN.						
Accuracy [%]	97.27	98.81	98.18	95	51.67	97.85
MCC [%]	96.15	98.25	97.79	89.08	43.98	85.05
Kappa [%]	23.90	41.87	42.13	94.29	99.63	93.29
Obs. Normal	2918	16	32	7	5	
Obs. Probe	20	2233	9	0	0	
Obs. DoS	21	8	2219	0	0	
Obs. R2L	34	2	0	198	1	
Obs. U2R	7	0	0	4	6	
$k = 1$ with $n = 10,000$, $n_{pca} = 16$ and CNN.						
Accuracy [%]	97.16	98.68	98.61	96.78	86.00	98.00
MCC [%]	96.30	98.33	98.22	89.69	55.24	87.56
Kappa [%]	24.00	41.91	41.92	94.30	99.56	93.77
Obs. Normal	2915	22	24	5	0	
Obs. Probe	17	2232	7	0	0	
Obs. DoS	19	6	2231	0	0	
Obs. R2L	35	2	1	198	1	
Obs. U2R	14	0	0	1	9	

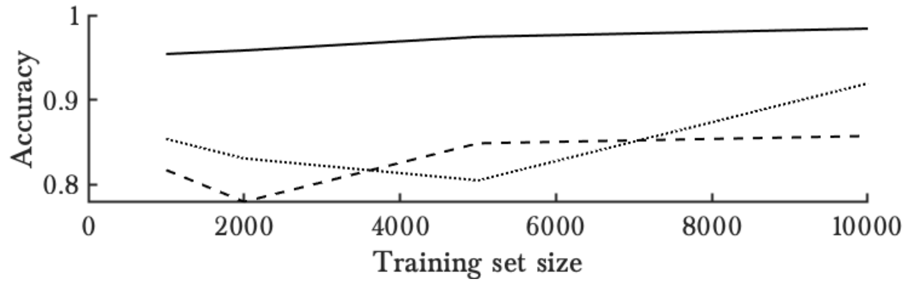
TABLE 4.12:]
Detailed results of the k-NN classification algorithm for $k = 1$ with condensed neighbours reduction and PCA reduction.

Model	Normal	Probe	DoS	R2L	U2R	Total
$k = 1$ with $n = 10,000$, χ^2 and CNN.						
Accuracy [%]	97.77	99.16	98.85	97.60	88.89	98.48
MCC [%]	96.98	99.03	98.91	92.61	45.63	86.63
Kappa [%]	23.66	41.82	41.97	94.35	99.44	95.24
Obs. Normal	2933	15	24	5	0	
Obs. Probe	12	2242	0	0	0	
Obs. DoS	7	2	2235	0	0	
Obs. R2L	22	2	2	203	1	
Obs. U2R	26	0	0	0	8	

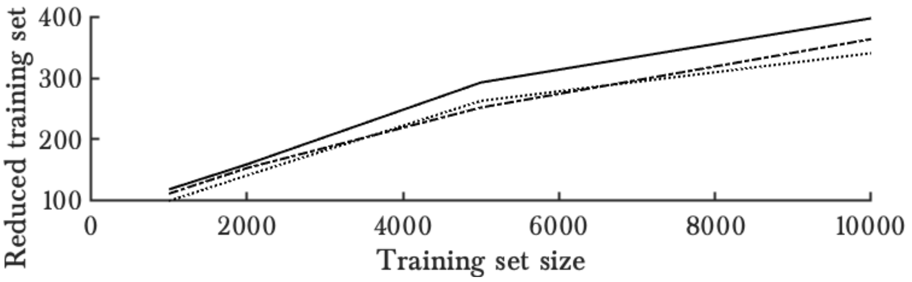
TABLE 4.13: Detailed results of the k-NN classification algorithm for $k = 1$ with condensed neighbours reduction and χ^2 feature selection.

4.4.4 Secure evaluation

We can now estimate the different models according to the computational cost, the round cost and the communication cost (figure 4.18). The accuracy is plotted here with the same value for each training set size using condensed nearest neighbours as it is not a parameter but a result of the reduction. Good performance have training set sizes after reduction ranging between approximately 300 and 500 for an original



(A) Accuracy with condensed nearest neighbours and χ^2 feature selection.



(B) Instance set size after reduction with condensed nearest neighbours and χ^2 feature selection.

FIGURE 4.16: Performance of the k-NN algorithm with χ^2 feature selection (33) and CNN reduction. The plain line is for $k = 1$, the dashed is $k = 2$ and the dotted is $k = 3$.

size of respectively 10,000 and 50,000. We benchmark the secure nearest neighbours algorithm with $n = 200$, $n = 500$ and $n = 1000$. The models tested are

- **(P)-1NN**: secure nearest neighbours evaluation with $k = 1$;
- **(P)-2NN**: secure nearest neighbours evaluation with $k = 2$;
- **(P)-3NN**: secure nearest neighbours evaluation with $k = 3$;
- **(P)PCA8-(P)-1NN**: secure nearest neighbours evaluation with secure PCA decomposition using 8 principal components;
- **(P)PCA16-(P)-1NN**: secure nearest neighbours evaluation with secure PCA decomposition using 16 principal components;
- χ^2**-(P)1NN**: secure nearest neighbours evaluation with χ^2 feature selection (33).

From the previous classification performance analysis, we know that the χ^2 does perform better for the same training set size after reduction. The reduction to 33 features also leads to a smaller distance evaluation which reduces the global evaluation time. The round communication costs are very similar as the largest contributor is the selection phase, which is the same. The secure nearest neighbours with χ^2 feature selection is thus as good or better in all indicators than the version without and should always be preferred.

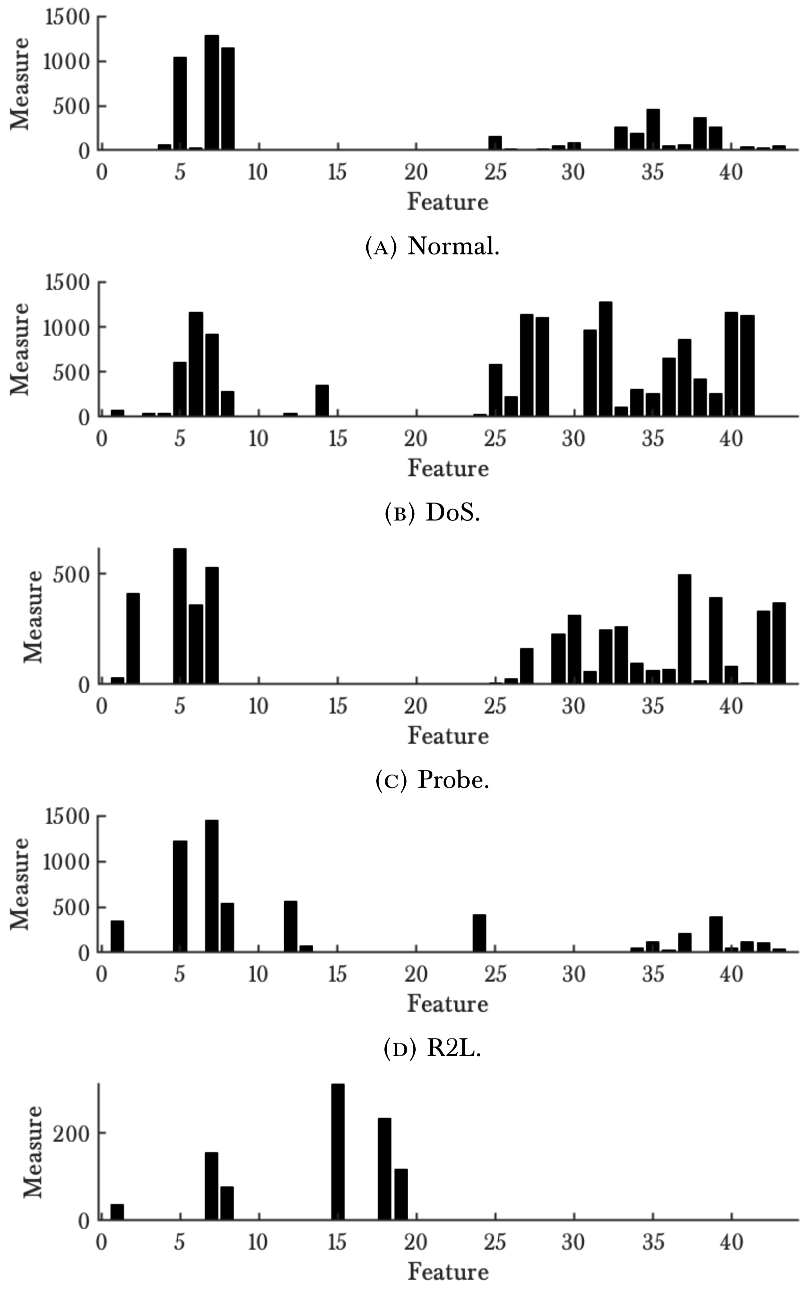


FIGURE 4.17: χ^2 measure of the k-NN algorithm for each feature and for each class.

The use of PCA decomposition however leads to a very significant decrease of the run time. Indeed, the extra cost of computing the decomposition is largely compensated by the reduction of features during the distance computation. The accuracy is not as good as the previous methods, nor is it significantly worse. The secure nearest neighbours should be preferred if a very small classification performance loss is accepted in order to divide the execution time almost by half.

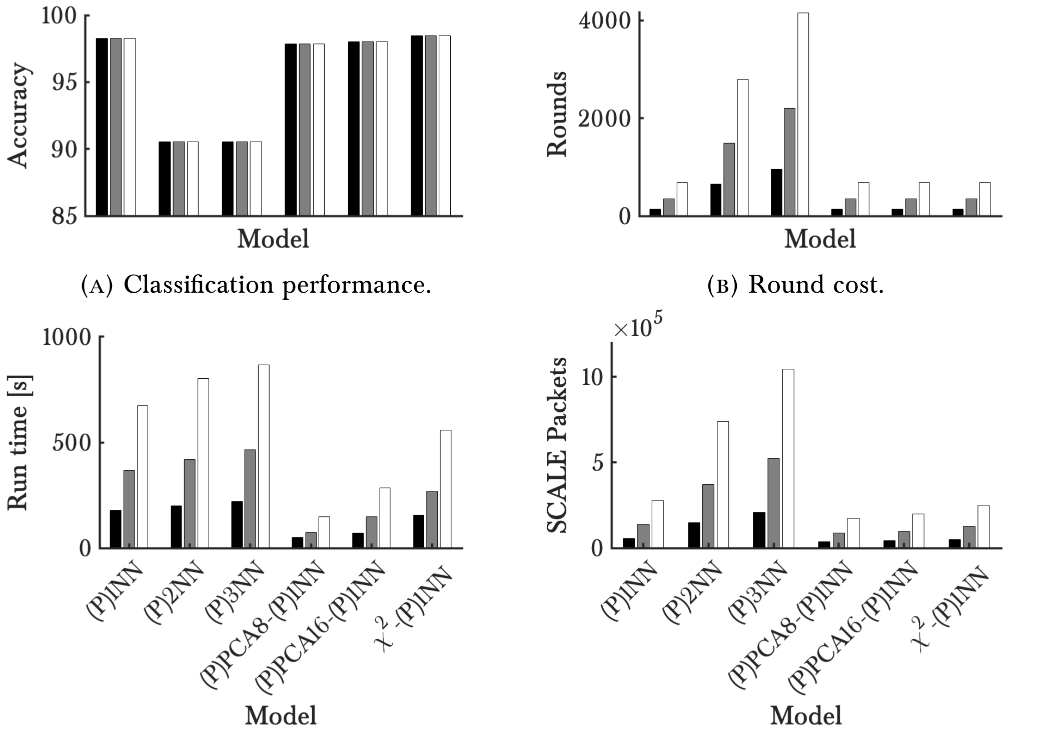


FIGURE 4.18: Comparison of different protocols for secure nearest neighbours evaluation with and without various feature size reduction methods. The black bars correspond to a training set size of $n = 200$, the grey to $n = 500$ and the white to $n = 1000$. The results correspond here to the secret evaluation of one unique query.

Chapter

5 Conclusion and future works

Privacy-friendly machine learning algorithms for intrusion detection systems are appealing for network defense and more generally to the protection of private information.

Secure linear support vector machines offer solutions at a very low cost and good accuracy (~ 95%), where nearest neighbors algorithm offer a higher accuracy (~ 98%), but at high cost. The use of condensed nearest neighbors allows to drastically diminish that cost to make the algorithm practically usable. As for now, non-linear support vector machines didn't prove to be competitive against nearest neighbors methods using MPC.

Various feature reduction methods can be used to reduce the evaluation costs even more, such as PCA or χ^2. When algorithms are fast, χ^2 is to be preferred as it requires no pre-process. However, when the algorithms are slower and the pre-process is proportionally less expensive, PCA reduction is to be preferred as it reduces the feature vector into less dimensions for the same accuracy.

This research raises several questions for further work. One of them is whether the proposed algorithm can be improved to make them even faster. For example, it could be possible to increase the rapidity of the RBFSVM multi-class models by avoiding computing the kernel function with the same support vectors appearing multiple times (in the different SVMs of the multi-class model). However, we could predict that this is would not lead to a significant gain as the support vectors are by essence of a support vector machine, the samples near the decision boundaries. Though, these decision boundaries should be different for each class. Hence, few redundancy is expected.

Another improvement could be using more efficient algorithms, e.g. for the minimum selection phase of the nearest neighbors. The quickselect algorithm has been proposed, but presents a lot of concerns about index leakage and thus a breach in the privacy. However, solutions exist to exchange two indices of an array in a secret manner [5].

In addition to algorithmic improvements, other dimensionality reduction algorithms exist that could maybe lead to further gain in execution cost, e.g. non-negative matrix factorization, kernel principal component analysis, linear discriminant analysis or canonical correlation analysis. Most of them have never been implemented in the (non-secure) case of intrusion detection systems. Kernel principal component analysis (KPCA) are an exception as they have been tested and deliver good results [36].

Unfortunately, they are evaluated in a very similar manner to RBFSVMs and thus we expect to encounter the same problem: the high cost of computing the kernel matrix.

Further work could also include the investigation of training models directly using MPC instead of using models separately trained. This could allow different data-base owners to aggregate their data in a secret manner and to train a model on it. This way, we should be able to gain much more of the performances of distributed intrusion detection systems. A first step is the use ensemble classifiers where each model is trained on its own and the results are then aggregated using a (weighted) majority vote. This just requires to train the weights of the majority vote in a secret manner. Furthermore, it should allow the models to be more resistant against attacks that try to exploit the output of the model against specific queries trying to reconstruct some information about the data-base. Ensemble models [2] would dilute the information obtained between different models.

Appendix

Appendix

A Implementations

All the code used can be found at `https://github.com/hdeplaen/masterthesis-src`.

A.1 MATLAB

The MATLAB implementation has been constructed as a toolbox. The toolbox is built around the notion of *experts*, which is an entire model as described in the thesis (one-against-all RBFSVMs or k-NN with consensed nearest neighbors). The different experts usable are:

- `'knn'`: normal nearest neighbors;
- `'kmeans-knn'`: nearest neighbors with k-means instance set reduction;
- `'knn-cnn'`: consensed nearest neighbors;
- `'knn-cnn-chi2'`: condensed nearest neighbors with χ^2 feature selection;
- `'par-svm'`: one-against-all (parallel) SVM multi-class model;
- `'seq-svm'`: tree-based (sequential) SVM multi-class model;
- `'par-svm'`: one-against-all (parallel) SVM multi-class model with χ^2 feature selection;
- `'seq-svm'`: tree-based (sequential) SVM multi-class model with χ^2 feature selection.

Here is a short list of the main functions that can be used:

- `load_data`: extraction of the data-set with conversion from categorical to numerical values (makes use of subroutines `to_numeric`, `cat2freq` and `cat2bin`);
- `normalize_data`: normalizes the data according to the values of the training set;
- `bagging`: creates different training and test sets according to section 4.2.3;

- `pca_reduction`: performs the PCA reduction to the specified number of components;
- `kmeans_clustering`: performs k-means clustering according to the training set;
- `train_expert`: generates and expert model with training if required;
- `eval_expert`: evaluates the expert according to the test set;
- `plot_perf`: computes and eventually plots the performance of the classifier on the test set;
- `export_expert`: exports the expert model for MPC-usage.

More information about each function (e.g. input and output arguments) can be found by typing `help function`, where `function` is the queried function.

Here follows a short example of the toolbox used to load, train, evaluate and export a condensed nearest neighbors expert with PCA reduction:

```matlab
%% PRELIMINARIES
k = 1;                  % number of nearest neighbors
n = 10000;              % number of elements in the training sets
num_bags = 5;           % number of experiments
disp_pca = false;       % don't plot the PCA componenets
expert = 'knn-cnn';     % type of expert used
data-set = 'nsl-kdd';   % data-set used

% preallocate the experiments results
corr = zeros(num_bags,5,5);
accm = zeros(num_bags,5);
mccm = zeros(num_bags,5);
kappam = zeros(num_bags,5);
acc = zeros(num_bags);
mcc = zeros(num_bags);
kappa = zeros(num_bags);
num_nb = zeros(num_bags);

%% GENERATE TRAINING AND TEST SETS
[trainX,trainY,testX,testY] = load_kdd(data_set,classes_red) ;
[BagTrainX,BagTrainY] = bagging(n(idxn), num_bags, trainX, trainY) ;
[BagTestX,BagTestY] = bagging(10000, num_bags, testX, testY) ;

% EXECUTE EACH EXPERIMENT
for idx_bag = 1:num_bags
    % extract the corresponding training set for this experiment
    locX = BagTrainX{idx_bag} ;
    locY = BagTrainY{idx_bag} ;

    % extract the corresponding test set for this experiment
    locXtest = BagTestX{idx_bag} ;
    locYtest = BagTestY{idx_bag} ;

    % normalize
    [locX,locY,locXtest,locYtest] = ...
        normalize_data(locX,locY,locXtest,locYtest) ;

    % PCA
    [trainX,testX] = pca_reduction(trainX,testX,n_pca,disp_pca) ;

    %% train and evaluate expert
    params_knn.k = k ;
    expert_knn = train_expert(locX,locY, expert, params_knn) ;
    eval_knn = eval_expert(expert_knn, locXtest, locYtest) ;
    export_expert(expert_knn, 'MyFirstExpert') ;

    [corr_, accm_, mccm_, kappam_, acc_, mcc_, kappa_] = ...
        plot_perf(eval_knn,locYtest) ;

    % store the results of each experiment
    corr(idx_bag,:,:) = corr_ ;
    accm(idx_bag,:) = accm_ ;
    mccm(idx_bag,:) = mccm_ ;
    kappam(idx_bag,:) = kappam_ ;
    acc(idx_bag) = acc_ ;
    mcc(idx_bag) = mcc_ ;
    kappa(idx_bag) = kappa_ ;
    num_nb(idx_bag) = expert_knn.num_nb ;
end

%% PLOT RESULTS
% mean of all experiments
corr = round(mean(corr(:,:,:),1));
accm = mean(accm(:,:),1);
mccm = mean(mccm(:,:),1);
kappam = mean(kappam(:,:),1);
acc = mean(acc(:),1);
mcc = mean(mcc(:),1);
kappa = mean(kappa(:),1);
num_nb = mean(num_nb(:),1);

% display the mean results
disp('k = 1') ;
disp('acc');
disp(squeeze(100*acc) ;
disp(squeeze(100*accm(:,:)));
disp('mcc') ;
disp(squeeze(100*mcc)) ;
disp(squeeze(100*mccm(:,:)));
disp('kappa') ;
disp(squeeze(100*kappa)) ;
disp(squeeze(100*kappam(:,:)));
disp('corr') ;
disp(squeeze(corr(:,:,:))) ;
```

A.2 SCALE-MAMBA

To compile and run the SCALE-MAMBA code, the framework first has to be installed. The installation and compilation are explained in the documentation [4]. The programs have to be used with training data produced by the MATLAB code and place in the `data` folder of each program. The following programs can be found:

- `knn_1`: Secure nearest neighbors evaluation with $k = 1$;
- `knn_chi2`: Secure nearest neighbors evaluation with $k = 1$ and χ^2 feature selection;
- `knn_1_pca`: Secure nearest neighbors evaluation with $k = 1$ and PCA reduction;
- `knn_n`: Secure nearest neighbors evaluation with $k > 1$;
- `svm_lin`: Secure linear support vector machines evaluation;
- `svm_lin_chi2`: Secure linear support vector machines with χ^2 feature selection;
- `svm_lin_pca`: Secure linear support vector machines with PCA reduction (the PCA or SVM evaluation can be done securely or in clear by commenting the corresponding lines);
- `svm_rbf`: Secure support vector machines with radial based kernel function;
- `svm_rbf_chi2`: Secure support vector machines with radial based kernel function with χ^2 feature selection;
- `svm_rbf_pca`: Secure support vector machines with radial based kernel function with PCA reduction.

Appendix

B Additional experimental results

B.1 Support Vector Machines

B.1.1 Linear SVMs

Model	Normal	Probe	DoS	R2L	U2R	Total
Tree 1 with $n = 2000$						
Accuracy [%]	87.67	96.36	96.13	94.30	46.67	92.81
MCC	86.34	95.69	91.36	72.66	35.29	70.86
Kappa	29.17	42.85	41.90	92.27	99.66	70.88
Obs. Normal	2630	38	175	145	12	
Obs. Probe	56	2172	20	5	0	
Obs. DoS	63	17	2167	6	1	
Obs. R2L	13	0	0	215	0	
Obs. U2R	0	0	0	5	4	
Tree 2 with $n = 2000$						
Accuracy [%]	90.55	96.22	95.37	91.93	44.44	91.96
MCC	87.67	96.01	91.27	78.55	34.90	77.45
Kappa	27.41	43.01	42.32	93.14	99.70	74.04
Obs. Normal	2717	27	161	86	9	
Obs. Probe	65	2169	17	2	0	
Obs. DoS	86	14	2150	4	0	
Obs. R2L	17	1	0	210	1	
Obs. U2R	0	0	0	5	4	
O-A-A with $n = 2000$						
Accuracy [%]	91.70	95.41	93.40	92.46	44.44	92.07
MCC	86.90	95.87	90.26	79.28	42.14	74.74
Kappa	26.47	43.43	43.22	93.16	99.75	75.22
Obs. Normal	2751	11	149	83	6	
Obs. Probe	88	2151	14	1	0	
Obs. DoS	127	17	2105	5	0	
Obs. R2L	17	0	0	211	0	
Obs. U2R	0	0	0	5	4	

Table B.1: Detailed results of the linear SVM classification algorithm for different multi-class models for $n = 2000$.

Model	Normal	Probe	DoS	R2L	U2R	Total
Tree 1 with $n = 30,000$						
Accuracy [%]	91.37	97.40	96.34	94.07	73.33	94.62
MCC [%]	89.00	96.03	93.15	85.20	67.38	86.15
Kappa [%]	27.06	42.34	42.25	93.59	99.66	83.19
Obs. Normal	2741	63	136	55	5	
Obs. Probe	53	2195	2	3	0	
Obs. DoS	76	5	2171	1	0	
Obs. R2L	13	0	0	213	0	
Obs. U2R	2	0	0	1	9	
Tree 2 with $n = 30,000$						
Accuracy [%]	92.56	97.11	96.18	93.27	66.67	95.12
MCC	89.84	96.82	92.39	87.16	73.46	89.12
Kappa	26.35	42.70	42.14	93.79	99.72	84.74
Obs. Normal	2777	29	151	42	2	
Obs. Probe	56	2189	9	0	0	
Obs. DoS	80	6	2168	0	0	
Obs. R2L	14	1	0	211	0	
Obs. U2R	0	0	0	4	8	
O-A-A with $n = 30,000$						
Accuracy [%]	93.03	96.67	96.53	94.60	70	94.62
MCC	90.13	96.84	92.81	88.31	77.52	87.93
Kappa	26.07	42.96	42.05	93.78	99.72	84.12
Obs. Normal	2791	18	150	40	1	
Obs. Probe	71	2179	4	0	0	
Obs. DoS	70	8	2176	0	0	
Obs. R2L	12	0	0	214	0	
Obs. U2R	0	0	0	3	8	

TABLE B.2: Detailed results of the linear SVM classification algorithm for different multi-class models for $n = 30,000$.

Model	Normal	Probe	DoS	R2L	U2R	Total
Tree 1 $n = 100,000$						
Accuracy [%]	93.19	97.77	95.68	35.75	10.77	93.96
MCC	87.08	96.69	92.77	50.96	18.95	69.29
Kappa	25.33	41.94	42.19	96.34	99.79	80.28
Obs. Normal	2796	48	133	22	1	
Obs. Probe	48	2214	2	0	0	
Obs. DoS	91	7	2167	0	0	
Obs. R2L	123	0	0	69	1	
Obs. U2R	11	0	0	1	1	
Tree 2 $n = 100,000$						
Accuracy [%]	93.51	97.35	96.74	80.41	36.92	95.41
MCC	90.34	97.31	92.79	82.31	54.36	87.31
Kappa	25.66	42.33	41.57	95.17	99.76	85.67
Obs. Normal	2805	20	150	24	0	
Obs. Probe	49	2205	11	0	0	
Obs. DoS	67	6	2191	1	0	
Obs. R2L	37	0	0	155	1	
Obs. U2R	6	0	0	2	5	
O-A-A $n = 100,000$						
Accuracy [%]	93.39	96.97	97.58	85.49	61.54	93.96
MCC	90.77	97.50	93.26	83.32	71.71	83.42
Kappa	25.82	42.60	41.19	94.93	99.71	84.87
Obs. Normal	2802	7	164	27	0	
Obs. Probe	67	2196	2	0	0	
Obs. DoS	45	4	2210	6	0	
Obs. R2L	27	0	0	165	1	
Obs. U2R	2	0	0	3	8	

TABLE B.3: Detailed results of the linear SVM classification algorithm for different multi-class models for $n = 100,000$.

B.1.2 Linear SVM with PCA decomposition

Model	Normal	Probe	DoS	R2L	U2R	Total
Tree with $n_{pca} = 8$						
Accuracy [%]	91.49	92.01	91.33	93.33	3.64	91.51
MCC [%]	87.21	91.53	87.34	70.16	$\emptyset$	$\emptyset$
Kappa [%]	26.55	43.84	43.22	93.69	99.84	71.11
Obs. Normal	2745	46	123	86	0	
Obs. Probe	93	2089	87	1	0	
Obs. DoS	104	42	2073	51	0	
Obs. R2L	12	0	0	168	0	
Obs. U2R	5	0	0	6	0	
O-A-A with $n_{pca} = 8$						
Accuracy [%]	87.50	91.54	94.28	95.33	12.73	90.75
MCC [%]	85.62	91.02	88.42	71.14	4.13	68.07
Kappa [%]	29.07	43.99	41.76	93.60	98.90	71.11
Obs. Normal	2625	50	137	127	61	
Obs. Probe	76	2078	112	2	1	
Obs. DoS	67	44	2140	11	9	
Obs. R2L	7	0	0	172	1	
Obs. U2R	2	0	0	8	1	

Table B.4: Detailed results of the linear SVM classification algorithm for different multi-class models with PCA decomposition and $n_{pca} = 8$ components kept. The first one is a tree-based model of order {Normal, DoS, Prob, R2L, U2R} and the second one a one-against-all model. Every result if the mean of 5 independent experiments.

Model	Normal	Probe	DoS	R2L	U2R	Total
Tree with $n_{pca} = 16$						
Accuracy [%]	91.33	94.94	94.19	96.11	33.33	93.24
MCC	87.13	95.35	90.77	79.68	$\emptyset$	$\emptyset$
Kappa	26.76	43.51	42.83	93.25	99.62	78.86
Obs. Normal	2740	12	150	89	8	
Obs. Probe	98	2142	16	1	0	
Obs. DoS	100	22	2125	8	1	
Obs. R2L	8	0	0	208	0	
Obs. U2R	6	0	0	4	5	
O-A-A with $n_{pca} = 16$						
Accuracy [%]	91.11	95.55	95.63	95.09	18.67	93.31
MCC	88.16	95.62	91.18	80.59	25.99	76.31
Kappa	27.08	43.23	42.09	93.42	99.74	80.29
Obs. Normal	2733	24	162	80	1	
Obs. Probe	72	2156	26	3	0	
Obs. DoS	80	15	2157	2	1	
Obs. R2L	10	0	1	205	0	
Obs. U2R	6	0	0	6	3	

Table B.5: Detailed results of the linear SVM classification algorithm for different multi-class models with PCA decomposition and $n_{pca} = 16$ components kept. The first one is a tree-based model of order {Normal, DoS, Prob, R2L, U2R} and the second one a one-against-all model. Every result if the mean of 5 independent experiments.

B.1.3 Radial function based kernel support vector machines

Model	Normal	Probe	DoS	R2L	U2R	Total
Tree 1 with $n = 15,000$						
Accuracy [%]	97.82	99.08	99.01	91.71	24.00	98.20
MCC [%]	96.62	99.03	98.54	87.47	32.25	82.78
Kappa [%]	23.65	42.23	42.14	93.71	99.69	94.39
Obs. Normal	2935	7	21	33	5	
Obs. Probe	18	2229	2	0	0	
Obs. DoS	19	2	2228	1	0	
Obs. R2L	16	1	1	215	1	
Obs. U2R	5	0	0	6	4	
Tree 2 with $n = 15,000$						
Accuracy [%]	97.98	98.25	99.20	91.45	28.00	98.08
MCC [%]	96.23	98.44	98.56	89.30	$\emptyset$	$\emptyset$
Kappa [%]	23.46	42.56	42.04	93.85	99.72	94.00
Obs. Normal	2939	8	24	26	2	
Obs. Probe	36	2211	4	0	0	
Obs. DoS	17	1	2232	0	0	
Obs. R2L	19	1	0	214	0	
Obs. U2R	7	0	0	3	4	
O-A-A with $n = 15,000$						
Accuracy [%]	98.33	99.40	99.15	92.14	24.00	98.55
MCC [%]	97.28	99.41	98.84	88.54	33.55	83.52
Kappa [%]	23.38	42.13	42.14	93.76	99.72	95.47
Obs. Normal	2950	4	15	30	1	
Obs. Probe	12	2237	1	0	0	
Obs. DoS	18	1	2231	0	0	
Obs. R2L	15	1	0	216	2	
Obs. U2R	5	0	1	5	4	

TABLE B.6: Detailed results of the RBF-SVM classification algorithm for different multi-class models for $n = 15,000$. The first one is a tree-based model of order {Normal, DoS, Prob, R2L, U2R} and the second one a one-against-all model. Every result if the mean of 5 independent experiments.

Model	Normal	Probe	DoS	R2L	U2R	Total
Tree 1 with $n = 50,000$						
Accuracy [%]	98.79	99.38	99.32	92.35	35.56	98.88
MCC [%]	97.82	99.42	98.98	91.23	48.81	87.25
Kappa [%]	23.00	41.75	41.68	94.69	99.82	96.46
Obs. Normal	2964	3	16	16	1	
Obs. Probe	13	2248	1	0	0	
Obs. DoS	13	2	2247	0	0	
Obs. R2L	15	0	0	188	1	
Obs. U2R	3	0	0	3	3	
Tree 2 with $n = 50,000$						
Accuracy [%]	98.93	99.02	99.27	91.76	40.00	98.80
MCC [%]	97.57	99.14	98.99	92.25	53.45	88.28
Kappa [%]	22.86	41.89	41.71	94.78	99.82	96.24
Obs. Normal	2968	5	15	12	1	
Obs. Probe	22	2240	1	0	0	
Obs. DoS	16	0	2245	0	0	
Obs. R2L	16	0	0	187	1	
Obs. U2R	3	0	0	2	4	
O-A-A with $n = 50,000$						
Accuracy [%]	98.80	99.59	99.37	91.86	37.78	98.95
MCC [%]	97.94	99.61	99.03	91.09	45.08	86.55
Kappa [%]	23.02	41.67	41.66	94.71	99.81	96.70
Obs. Normal	2964	3	16	16	1	
Obs. Probe	8	2253	1	0	0	
Obs. DoS	13	1	2248	0	0	
Obs. R2L	15	0	0	187	1	
Obs. U2R	3	0	0	2	3	

Table B.7: Detailed results of the RBF-SVM classification algorithm for different multi-class models for $n = 50,000$. The first one is a tree-based model of order {Normal, DoS, Prob, R2L, U2R} and the second one a one-against-all model. Every result if the mean of 5 independent experiments.

B.2 Nearest neighbours

Model	Normal	Probe	DoS	R2L	U2R	Total
$k = 1$ with $n = 10,000$						
Accuracy [%]	98.17	99.45	98.80	98.03	43.33	98.66
MCC [%]	97.49	99.19	98.63	92.76	39.68	85.55
Kappa [%]	23.48	41.80	42.04	94.07	99.74	95.81
Obs. Normal	2945	11	15	25	4	
Obs. Probe	10	2246	2	1	0	
Obs. DoS	23	3	2231	1	1	
Obs. R2L	2	0	0	214	2	
Obs. U2R	3	0	0	2	4	
$k = 2$ with $n = 10,000$						
Accuracy [%]	98.26	99.37	98.26	95.93	30	98.42
MCC [%]	97.08	98.86	98.44	90.31	38.74	84.69
Kappa [%]	23.24	41.49	42.03	94.82	99.75	95.07
Obs. Normal	2948	17	9	24	2	
Obs. Probe	12	2252	1	1	0	
Obs. DoS	34	5	2227	1	0	
Obs. R2L	6	0	0	181	1	
Obs. U2R	3	0	0	5	4	
$k = 3$ with $n = 10,000$						
Accuracy [%]	97.05	98.90	98.80	97.81	55.00	98.08
MCC [%]	96.36	98.63	98.31	89.11	39.73	84.43
Kappa [%]	24.10	41.88	41.87	94.06	99.69	94.00
Obs. Normal	2912	15	22	43	10	
Obs. Probe	18	2236	5	1	0	
Obs. DoS	22	4	2234	1	0	
Obs. R2L	3	0	0	205	2	
Obs. U2R	3	0	0	1	4	

TABLE B.8: Detailed results of the k-NN classification algorithm for two different values of the number of neighbours k and for a small training data-set.

Model	Normal	Probe	DoS	R2L	U2R	Total
$k = 1$ with $n = 100,000$						
Accuracy [%]	99.56	99.82	99.63	93.81	58.33	99.48
MCC [%]	99.00	99.73	99.57	95.39	62.16	91.17
Kappa [%]	22.61	41.51	41.58	94.88	99.85	98.36
Obs. Normal	2987	4	4	5	1	
Obs. Probe	2	2260	2	0	0	
Obs. DoS	8	0	2256	0	0	
Obs. R2L	12	0	0	190	1	
Obs. U2R	2	0	0	1	4	
$k = 2$ with $n = 100,000$						
Accuracy [%]	99.19	99.92	99.57	94.32	48.89	99.33
MCC [%]	98.64	99.72	99.35	95.07	65.47	91.65
Kappa [%]	22.86	41.52	41.64	94.73	99.82	97.90
Obs. Normal	2976	7	11	7	0	
Obs. Probe	2	2260	0	0	0	
Obs. DoS	9	1	2252	0	0	
Obs. R2L	12	0	0	194	0	
Obs. U2R	4	0	0	1	4	
$k = 3$ with $n = 100,000$						
Accuracy [%]	99.32	99.77	99.66	89.85	40.83	99.22
MCC [%]	98.48	99.65	99.37	93.37	54.52	89.08
Kappa [%]	22.71	41.52	41.52	95.11	99.77	97.56
Obs. Normal	2980	5	10	4	1	
Obs. Probe	4	2259	2	0	0	
Obs. DoS	8	0	2256	0	0	
Obs. R2L	19	1	0	177	0	
Obs. U2R	6	0	1	1	5	

TABLE B.9: Detailed results of the k-NN classification algorithm for two different values of the number of neighbours k and a big training data-set.

B.2.1 k-means

Model	Normal	Probe	DoS	R2L	U2R	Total
$k = 1$ with $n = 100,000$						
Accuracy [%]	95.19	99.29	98.78	96.92	76	97.45
MCC [%]	94.88	97.05	98.29	89.79	74.85	90.97
Kappa [%]	25.09	41.02	41.67	94.85	99.60	92.02
Obs. Normal	2856	12	25	5	3	
Obs. Probe	84	2251	2	0	0	
Obs. DoS	23	4	2239	0	0	
Obs. R2L	34	1	0	179	0	
Obs. U2R	4	0	0	0	11	
$k = 2$ with $n = 100,000$						
Accuracy [%]	95.03	99.26	98.28	97.82	55.00	97.27
MCC [%]	94.59	97.16	97.86	89.90	33.75	82.65
Kappa [%]	25.26	41.32	42.08	93.91	99.82	91.48
Obs. Normal	2851	13	34	4	1	
Obs. Probe	73	2243	5	0	0	
Obs. DoS	27	3	2221	0	0	
Obs. R2L	43	1	0	211	1	
Obs. U2R	6	0	0	1	2	
$k = 3$ with $n = 100,000$						
Accuracy [%]	86.61	98.75	98.54	98.14	72.22	93.94
MCC [%]	87.98	93.01	96.85	82.97	61.41	84.44
Kappa [%]	29.48	40.32	41.77	93.22	99.72	81.06
Obs. Normal	2598	22	25	2	2	
Obs. Probe	246	2229	4	0	0	
Obs. DoS	65	5	2224	0	0	
Obs. R2L	87	1	4	216	1	
Obs. U2R	4	0	0	2	7	

TABLE B.10: Detailed results of the k-NN classification algorithm for two different values of the number of neighbours k and k-means data reduction.

B.2.2 Condensed nearest neighbors

Model	Normal	Probe	DoS	R2L	U2R	Total
$k = 1$ with $n = 10,000$						
Accuracy [%]	97.63	99.11	98.26	98.16	80	98.25
MCC [%]	96.73	99.02	98.02	90.27	59.45	88.70
Kappa [%]	23.74	41.94	42.19	93.92	99.70	94.52
Obs. Normal	2929	14	31	3	1	
Obs. Probe	9	2238	1	1	0	
Obs. DoS	20	4	2219	0	0	
Obs. R2L	36	1	4	213	1	
Obs. U2R	6	0	3	0	6	
$k = 2$ with $n = 10,000$						
Accuracy [%]	93.67	94.48	78.75	68.56	36.67	88.81
MCC [%]	85.64	84.77	82.47	78.84	36.00	73.54
Kappa [%]	24.58	40.61	48.94	95.17	99.86	65.03
Obs. Normal	2810	44	238	66	2	
Obs. Probe	172	2135	241	0	0	
Obs. DoS	6	81	1780	0	0	
Obs. R2L	10	0	1	147	2	
Obs. U2R	2	0	0	1	2	
$k = 3$ with $n = 10,000$						
Accuracy [%]	94.37	90.02	88.40	86.43	36.47	91.01
MCC [%]	90.39	85.97	85.30	85.11	51.17	79.59
Kappa [%]	25.04	43.77	44.66	94.26	99.67	71.91
Obs. Normal	2831	41	106	29	10	
Obs. Probe	101	2029	155	0	0	
Obs. DoS	35	182	1993	0	0	
Obs. R2L	32	2	1	191	0	
Obs. U2R	1	0	0	1	6	

TABLE B.11: Detailed results of the k-NN classification algorithm for $k = 1, 2, 3$ with condensed neighbours reduction.

B.2.3 CNN with PCA

Model	Normal	Probe	DoS	R2L	U2R	Total
$k = 1$ with $n = 10,000$ and $n_{pca} = 8$						
Accuracy [%]	97.27	98.81	98.18	95	51.67	97.85
MCC [%]	96.15	98.25	97.79	89.08	43.98	85.05
Kappa [%]	23.90	41.87	42.13	94.29	99.63	93.29
Obs. Normal	2918	16	32	7	5	
Obs. Probe	20	2233	9	0	0	
Obs. DoS	21	8	2219	0	0	
Obs. R2L	34	2	0	198	1	
Obs. U2R	7	0	0	4	6	
$k = 2$ with $n = 10,000$ and $n_{pca} = 8$						
Accuracy [%]	92.44	99.45	66.28	68.14	4	86.03
MCC [%]	84.16	82.18	75.89	74.38	$\emptyset$	$\emptyset$
Kappa [%]	25.45	36.97	53.77	94.51	99.85	56.35
Obs. Normal	2773	9	275	72	6	
Obs. Probe	188	2241	482	1	0	
Obs. DoS	6	3	1493	0	0	
Obs. R2L	33	1	2	157	4	
Obs. U2R	1	0	0	1	0	
$k = 3$ with $n = 10,000$ and $n_{pca} = 8$						
Accuracy [%]	95.82	89.26	81.71	86.70	16	89.43
MCC [%]	91.91	81.79	79.87	81.48	19.28	70.87
Kappa [%]	24.22	43.09	46.82	94.36	99.79	66.98
Obs. Normal	2875	59	82	26	6	
Obs. Probe	2875	59	82	26	6	
Obs. DoS	35	181	1847	0	0	
Obs. R2L	49	2	1	179	2	
Obs. U2R	4	0	0	1	2	

TABLE B.12: Detailed results of the k-NN classification algorithm for $k = 1, 2, 3$ with condensed neighbours reduction and PCA decomposition ($n_{pca} = 8$).

Model	Normal	Probe	DoS	R2L	U2R	Total
$k = 1$ with $n = 10,000$ and $n_{pca} = 16$						
Accuracy [%]	97.16	98.68	98.61	96.78	86	98.00
MCC [%]	96.30	98.33	98.22	89.69	55.24	87.56
Kappa [%]	24.00	41.91	41.92	94.30	99.56	93.77
Obs. Normal	2915	22	24	5	0	
Obs. Probe	17	2232	7	0	0	
Obs. DoS	19	6	2231	0	0	
Obs. R2L	35	2	1	198	1	
Obs. U2R	14	0	0	1	9	
$k = 2$ with $n = 10,000$ and $n_{pca} = 16$						
Accuracy [%]	90.79	96.59	71.94	75.38	17.50	86.50
MCC [%]	85.92	80.50	77.43	78.06	$\emptyset$	$\emptyset$
Kappa [%]	26.70	37.87	50.78	94.95	99.86	57.81
Obs. Normal	2724	16	179	48	4	
Obs. Probe	202	2187	453	0	0	
Obs. DoS	37	59	1629	0	0	
Obs. R2L	36	3	2	150	2	
Obs. U2R	1	0	0	0	1	
$k = 3$ with $n = 10,000$ and $n_{pca} = 16$						
Accuracy [%]	92.27	92.52	94.92	83.77	12.73	92.79
MCC [%]	90.67	88.72	90.60	80.12	15.14	73.05
Kappa [%]	26.51	42.91	42.02	94.88	99.74	77.48
Obs. Normal	2768	24	51	30	6	
Obs. Probe	140	2096	56	0	1	
Obs. DoS	51	143	2151	0	0	
Obs. R2L	35	2	7	160	3	
Obs. U2R	6	0	1	1	1	

TABLE B.13: Detailed results of the k-NN classification algorithm for $k = 1, 2, 3$ with condensed neighbours reduction and PCA decomposition ($n_{pca} = 16$).

B.2.4 CNN with χ^2 feature selection

Model	Normal	Probe	DoS	R2L	U2R	Total
$k = 1$ with $n = 10,000$, χ^2 and CNN.						
Accuracy [%]	97.77	99.16	98.85	97.60	88.89	98.48
MCC [%]	96.98	99.03	98.91	92.61	45.63	86.63
Kappa [%]	23.66	41.82	41.97	94.35	99.44	95.24
Obs. Normal	2933	15	24	5	0	
Obs. Probe	12	2242	0	0	0	
Obs. DoS	7	2	2235	0	0	
Obs. R2L	22	2	2	203	1	
Obs. U2R	26	0	0	0	8	
$k = 2$ with $n = 10,000$, χ^2 and CNN.						
Accuracy [%]	96.80	99.69	58.14	79.68	7.69	85.75
MCC [%]	89.53	79.61	70.04	84.74	19.56	68.70
Kappa [%]	23.02	35.61	56.12	95.48	99.81	55.47
Obs. Normal	2904	2	251	38	7	
Obs. Probe	82	2260	698	0	0	
Obs. DoS	3	5	1318	0	0	
Obs. R2L	10	0	0	149	5	
Obs. U2R	1	0	0	0	1	
$k = 3$ with $n = 10,000$, χ^2 and CNN.						
Accuracy [%]	94.83	84.84	96.02	86.63	54.55	91.99
MCC [%]	93.37	86.87	86.26	77.55	46.62	78.13
Kappa [%]	25.22	46.90	40.33	94.22	99.66	74.96
Obs. Normal	2845	9	50	27	2	
Obs. Probe	37	1919	36	0	0	
Obs. DoS	53	324	2172	0	0	
Obs. R2L	56	10	4	175	3	
Obs. U2R	9	0	0	0	6	

TABLE B.14: Detailed results of the k-NN classification algorithm with condensed neighbours reduction and χ^2 (33) feature selection.

Model	Normal	Probe	DoS	R2L	U2R	Total
$k = 1$ with $n = 10,000$, χ^2 and CNN.						
Accuracy [%]	91.23	93.80	93.66	90.54	87.50	97.87
MCC [%]	88.48	93.96	92.37	67.63	27.50	83.44
Kappa [%]	27.06	43.81	43.51	92.25	98.86	93.34
Obs. Normal	2737	55	91	12	1	
Obs. Probe	45	2117	7	0	0	
Obs. DoS	68	32	2114	0	0	
Obs. R2L	106	44	34	201	0	
Obs. U2R	44	9	11	9	7	
$k = 2$ with $n = 10,000$, χ^2 and CNN.						
Accuracy [%]	66.67	98.58	65.00	53.68	66.67	88.21
MCC [%]	53.49	74.83	73.74	63.41	10.96	72.02
Kappa [%]	37.25	35.09	54.09	94.98	97.21	63.14
Obs. Normal	2000	17	613	54	0	
Obs. Probe	841	2222	118	3	0	
Obs. DoS	23	6	1465	12	0	
Obs. R2L	21	0	14	124	2	
Obs. U2R	115	9	44	38	4	
$k = 3$ with $n = 10,000$, χ^2 and CNN.						
Accuracy [%]	91.27	4.60	85.09	80.09	0	76.65
MCC [%]	75.19	10.80	47.24	62.94	0.35	60.45
Kappa [%]	24.62	69.50	34.40	93.11	98.92	27.07
Obs. Normal	2738	412	249	30	1	
Obs. Probe	58	104	5	0	0	
Obs. DoS	94	1722	1923	2	0	
Obs. R2L	70	17	64	169	9	
Obs. U2R	40	5	19	10	0	

TABLE B.15: Detailed results of the k-NN classification algorithm with condensed neighbours reduction and χ^2 (20) feature selection.

Appendix

C

The Quickselect algorithm

Hoare's *quickselect* algorithm [40] is able to find the k^{th} smallest — alternatively biggest — elements of a list in best case scenario $\mathcal{O}(n_t)$ and worst case scenario $\mathcal{O}(n_t \log n_t)$ where n_t is the size of the training set. The practice shows that the algorithm tends to obtain a complexity closer to the best case than the worst case scenario. Using this algorithm, one could hope to obtain a k-NN computational complexity of $\mathcal{O}(n_t n_f)$ where n_f is the feature set size.

It uses an auxiliary subroutine that reorganizes a given partition — specified by two indices — into elements smaller than a given pivot point left of it and bigger, right of it. The main algorithm then makes recursive calls to that same subroutine until the pivot ends at the k^{th} position. The algorithm and its subroutine are given at 22 and 21 in clear.

For the algorithm to run using MPC, the comparisons of the main routine have to be revealed in order to decide whether or not make a recursive call. The indices i_a and i_b should also be revealed in order to perform the loop of the subroutine. We can however avoid revealing the results of the comparisons of the subroutine by using a secure conditional swap [5].

This algorithm has been implemented but the compilation results in an infinite loop. As the MAMBA language first develops all the python code, it develops the function containing the recursive call the that same function and adds it to the SCALE code. It does so with the new call and adds the new recursive call to the SCALE code. This goes on forever. Classical loops are also fully developed. The only way to really execute a loop is to use certain specific calls to specially developed routines that allow real loops on the SCALE code. Unfortunately, no such routines exist for recursive calls.

The revelation of the comparisons needed to make the (eventual) recursive calls, i_a and i_b would not guarantee the privacy of the algorithm as much information about the relative order of the distances would be revealed. An solution could be to scramble the training set before each evaluation [63], but at what costs?

Furthermore, the interest for this algorithm has to be nuanced in the case of intrusion detection systems as $k = 1$ provides the best results and can already be theoretically achieved in $\mathcal{O}(n_t)$.

```
Data: list $\{y_i\}$, partition start index $i_a$, partition end index $i_b$, pivot index $i_p$
Result: new pivot index $j_p$, sorted list $\{y_j\}$ of $\{y_{i_a}, \dots, y_{i_b}\}$ where
        $\forall j < j_p : y_j < y_{i_p}$, $\forall j > j_p : y_j > y_{i_p}$ and $y_{j_p} = y_{i_p}$
function partition($\{y_i\}$, $i_a$, $i_b$, $i_p$):
    $p \leftarrow y_{i_p}$
    $l \leftarrow a$
    for $i = i_a$ to $i_b - 1$ do
        if $y_i < p$ then
            swap $y_i$ and $y_k$
            $k \leftarrow k + 1$
        end
        swap $y_b$ and $y_k$
    end
    $j_p \leftarrow l$
    return $j_p$, $\{y_j\}$
```

Algorithm 21: Quickselect's partition subroutine

```
Data: list $\{y_i\}$, partition start index $i_a$, partition end index $i_b$, number of
      smallest elements wanted $k$
Result: new list $\{y_j\}$ containing $k$ smallest elements
function select($\{y_i\}$, $i_a$, $i_b$, $k$):
    if $i_a = i_b$ then
        return $\{y_1, \dots, y_{i_a}\}$
    else
        assign random $l \in [i_a, i_b]$
        $l$, $\{y_j\} \leftarrow$ partition($\{y_i\}$, $i_a$, $i_b$, $l$)
        if $k = l$ then
            return $\{y_1, \dots, y_k\}$
        else if $k < l$ then
            return select($\{y_i\}$, $i_a$, $l + 1$, $k$)
        else
            return select($\{y_i\}$, $l + 1$, $i_b$, $k$)
    end
```

Algorithm 22: Quickselect's main routine

Bibliography

[1] A. A. Aburomman and M. B. Ibne Reaz, „A novel SVM-kNN-PSO ensemble method for intrusion detection system”, *Applied Soft Computing*, vol. 38, pp. 360–372, Jan. 2016, ISSN: 15684946.

[2] A. A. Aburomman and M. B. I. Reaz, „A survey of intrusion detection systems based on ensemble and hybrid classifiers”, *Computers and Security*, vol. 65, pp. 135–152, 2017, ISSN: 01674048.

[3] Akashdeep, I. Manzoor, and N. Kumar, „A feature reduced intrusion detection system using ANN classifier”, *Expert Systems with Applications*, vol. 88, pp. 249–257, 2017, ISSN: 09574174.

[4] A. Aly, M. Keller, E. Orsini, D. Rotaru, N. P. Smart, and T. Wood, „SCALE and MAMBA Documentation”, pp. 1–101, 2018.

[5] A. Aly and M. Van Vyve, „Securely solving classical network flow problems”, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8949, pp. 205–221, 2014, ISSN: 16113349.

[6] J. Amudhavel, V. Brindha, B. Anantharaj, P. Karthikeyan, B. Bhuvaneswari, M. Vasanthi, D. Nivetha, and D. Vinodha, „A survey on Intrusion Detection System: State of the art review”, *Indian Journal of Science and Technology*, vol. 9, no. 11, pp. 1–9, 2016, ISSN: 09745645.

[7] F. Angiulli, „Fast condensed nearest neighbor rule”, *Proceedings of the 22nd international conference on Machine learning - ICML '05*, pp. 25–32, 2005, ISSN: 1595931805.

[8] D. Arthur and S. Vassilvitskii, „Worst-case and smoothed analysis of the ICP algorithm, with an application to the k-means method”, *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pp. 153–164, 2006, ISSN: 00295515.

[9] J. Bar-Ilan and D. Beaver, „Non-cryptographic fault-tolerant computing in constant number of rounds of interaction”, in *Proceedings of the eighth annual ACM Symposium on Principles of distributed computing - PODC '89*, New York, New York, USA: ACM Press, 1989, pp. 201–209, ISBN: 0897913264.

[10] A. Barnett, J. Santokhi, M. Simpson, N. P. Smart, C. Stainton-Bygrave, S. Vivek, and A. Waller, „Image Classification using non-linear Support Vector Machines on Encrypted Data", *IACR Cryptology ePrint Archive*, p. 857, 2017.

[11] D. Beaver, S. Micali, and P. Rogaway, „The round complexity of secure protocols", in *Proceedings of the twenty-second annual ACM symposium on Theory of computing - STOC '90*, New York, New York, USA: ACM Press, 1990, pp. 503–513, ISBN: 0897913612.

[12] R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias, „Semi-homomorphic encryption and multiparty computation", *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6632 LNCS, pp. 169–188, 2011, ISSN: 03029743.

[13] S. Bhatt, P. K. Manadhata, and L. Zomlot, „The Operational Role of Security Information and Event Management Systems", *IEEE Security & Privacy*, vol. 12, no. 5, pp. 35–41, Sep. 2014, ISSN: 1540-7993.

[14] F. Blom, *A performance evaluation of secure comparison protocols in the two-party semi-honest model [MSc Thesis]*. 2014.

[15] D. Bogdanov, „Sharemind: programmable secure computations with practical applications", *PhD Thesis, Institute of Computer Science, Faculty of Mathematics and Computer Science, University of Tartu, Estonia*, 2013.

[16] D. Bogdanov, L. Kamm, S. Laur, and V. Sokk, „Rmind: A tool for cryptographically secure statistical analysis", *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 3, pp. 481–495, 2018, ISSN: 15455971.

[17] D. Bogdanov, P. Laud, and J. Randmets, „Domain-Polymorphic Programming of", *Proceedings of the Ninth Workshop on Programming Languages and Analysis for Security*, pp. 1–24, 2014.

[18] D. Bogdanov, S. Laur, and J. Willemson, „Sharemind: A framework for fast privacy-preserving computations", *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5283 LNCS, pp. 192–206, 2008, ISSN: 03029743.

[19] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos, „SEPIA: Privacy-preserving Aggregation of Multi-domain Network Events and Statistics", *Proceedings of the 19th USENIX Conference on Security - Security '10*, p. 15, 2010.

[20] O. Catrina and S. De Hoogh, „Improved primitives for secure multiparty integer computation", *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6280 LNCS, pp. 182–199, 2010, ISSN: 03029743.

[21] O. Catrina and A. Saxena, „Secure computation with fixed-point numbers", *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6052 LNCS, pp. 35–50, 2010, ISSN: 03029743.

[22] S. Chebrolu, A. Abraham, and J. P. Thomas, „Feature deduction and ensemble design of intrusion detection systems", *Computers and Security*, vol. 24, no. 4, pp. 295–307, 2005, ISSN: 01674048.

[23] C. M. Chen, D. J. Guan, Y. Z. Huang, and Y. H. Ou, „Anomaly network intrusion detection using Hidden Markov Model", *International Journal of Innovative Computing, Information and Control*, vol. 12, no. 2, pp. 569–580, 2016, ISSN: 13494198.

[24] S. B. Cho and H. J. Park, „Efficient anomaly detection by modeling privilege flows using hidden Markov model", *Computers and Security*, vol. 22, no. 1, pp. 45–55, 2003, ISSN: 01674048.

[25] J. Cohen, „A Coefficient of Agreement for Nominal Scales", *Educational and Psychological Measurement*, vol. 20, no. 1, pp. 37–46, 1960.

[26] L. Dali, A. Bentajer, E. Abdelmajid, K. Abouelmehdi, H. Elsayed, E. Fatiha, and A. Beni-Hssane, „A survey of intrusion detection system", *2015 2nd World Symposium on Web Applications and Networking, WSWAN 2015*, 2015, ISSN: 01674048.

[27] I. Damgård, M. Geisler, M. Krøigaard, and J. B. Nielsen, „Asynchronous multiparty computation: Theory and implementation", *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5443, pp. 160–179, 2009, ISSN: 03029743.

[28] I. Damgård, M. Keller, and E. Larraia, „Practical Covertly Secure MPC for Dishonest Majority-or: Breaking the SPDZ Limits.", *IACR Cryptology ...*, pp. 1–18, 2012.

[29] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, „Multiparty computation from somewhat homomorphic encryption", *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7417 LNCS, pp. 643–662, 2012, ISSN: 03029743.

[30] D. Demmler, T. Schneider, and M. Zohner, „ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation", *Proceedings 2015 Network and Distributed System Security Symposium*, no. February, pp. 8–11, 2015, ISSN: 0018-5787.

[31] L. Dhanabal and D. S. P. Shantharajah, „A Study On NSL-KDD Dataset For Intrusion Detection System Based On Classification Algorithms", *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 4, no. 6, pp. 446–452, 2015, ISSN: 2319-5940.

[32] D. Dheeru and E. Karra Taniskidou, *UCI Machine Learning Repository*, 2017.

[33] N. Dowlin, R. Gilad-Bachrach, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, „CryptoNets: Applying neural networks to Encrypted data with high throughput and accuracy - Microsoft research", *International Conference on Machine Learning, A**, vol. 48, pp. 1–12, 2016, ISSN: 1938-7228.

[34] C. Dwork, „Differential Privacy: A Survey of Results", in *Lecture Notes in Computer Science*, vol. Theory and, Springer Verlag, 2008, pp. 1–19.

[35] H. F. Eid, A. Darwish, A. Ella Hassanien, and A. Abraham, „Principle components analysis and support vector machine based Intrusion Detection System", *Proceedings of the 2010 10th International Conference on Intelligent Systems Design and Applications, ISDA'10*, pp. 363–367, 2010, ISSN: 0888-5885.

[36] Z. Elkhadir, K. Chougdali, and M. Benattou, „Intrusion detection system using PCA and kernel PCA methods", *Lecture Notes in Electrical Engineering*, vol. 381, no. February, pp. 489–497, 2016, ISSN: 18761119.

[37] S. Even, O. Goldreich, and A. Lempel, „A randomized protocol for signing contracts", *Communications of the ACM*, vol. 28, no. 6, pp. 637–647, 1985, ISSN: 00010782.

[38] N. Farnaaz and M. A. Jabbar, „Random Forest Modeling for Network Intrusion Detection System", *Procedia Computer Science*, vol. 89, pp. 213–217, 2016, ISSN: 18770509.

[39] O. Goldreich, S. Micali, and A. Wigderson, „How to play ANY mental game", in *Proceedings of the nineteenth annual ACM conference on Theory of computing - STOC '87*, vol. 106, New York, New York, USA: ACM Press, 1987, pp. 218–229, ISBN: 0-89791-221-7.

[40] C. A. R. Hoare, „Algorithm 65: Find", *Commun. ACM*, vol. 4, no. 7, pp. 321–322, Jul. 1961, ISSN: 0001-0782.

[41] H. Hu, J. Xu, C. Ren, and B. Choi, „Processing private queries over untrusted data cloud through privacy homomorphism", *Proceedings - International Conference on Data Engineering*, pp. 601–612, 2011, ISSN: 10844627.

[42] S. T. Ikram and A. K. Cherukuri, „Improving accuracy of intrusion detection model using PCA and optimized SVM", *Journal of Computing and Information Technology*, vol. 24, no. 2, pp. 133–148, 2016, ISSN: 13301136.

[43] K. Ilgun, R. A. Kemmerer, and P. A. Porras, „State Transition Analysis : A Rule-Based Intrusion Detection Approach", vol. 21, no. 3, pp. 181–199, 1995.

[44] Y. Ishai and E. Kushilevitz, „Private simultaneous messages protocols with applications", in *Proceedings of the Fifth Israeli Symposium on Theory of Computing and Systems*, IEEE Comput. Soc, 1997, pp. 174–183, ISBN: 0-8186-8037-7.

[45] R. Jagomägis, „SecreC : a Privacy-Aware Programming Language with Applications in Data Mining", no. May, 2010.

[46] L. Khan, M. Awad, and B. Thuraisingham, „A new intrusion detection system using support vector machines and hierarchical clustering", *VLDB Journal*, vol. 16, no. 4, pp. 507–521, 2007, ISSN: 10668888.

[47] F. Kuang, W. Xu, and S. Zhang, „A novel hybrid KPCA and SVM with GA model for intrusion detection", *Applied Soft Computing Journal*, vol. 18, pp. 178–184, 2014, ISSN: 15684946.

[48] S. Laur, H. Lipmaa, and T. Mielikäinen, „Cryptographically private support vector machines", *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '06*, p. 618, 2006, ISSN: 16182642.

[49] H.-j. Liao, C.-h. R. Lin, Y.-c. Lin, and K.-y. Tung, „Intrusion detection system : A comprehensive review", *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16–24, 2013, ISSN: 1084-8045.

[50] E. Makri, D. Rotaru, N. P. Smart, and F. Vercauteren, „PICS: Private Image Classification with SVM", 2017.

[51] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella, „Fairplay - A Secure Two-Party Computation System", *Proceedings of the 13th Conference on USENIX Security Symposium*, pp. 20–37, 2004.

[52] B. W. Matthews, „Comparison of the predicted and observed secondary structure of T4 phage lysozyme", *BBA - Protein Structure*, vol. 405, no. 2, pp. 442–451, 1975, ISSN: 00052795.

[53] G. Meeragandhi and S. Shesh, „Effective Network Intrusion Detection using Classifiers Decision Trees and Decision rules", *Network*, vol. 692, pp. 686–692, 2010.

[54] Ming Tian, Song-can Chen, Yi Zhuang, and Jia Liu, „Using statistical analysis and support vector machine classification to detect complicated attacks", in *Proceedings of 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.04EX826)*, IEEE, 2004, pp. 2747–2752, ISBN: 0-7803-8403-2.

[55] B. Mukherjee, L. Heberlein, and K. Levitt, „Network intrusion detection", *IEEE Network*, vol. 8, no. 3, pp. 26–41, 1994, ISSN: 0890-8044.

[56] S. Mukkamala, G. Janoski, and A. Sung, „Intrusion detection using neural networks and support vector machines", in *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No.02CH37290)*, vol. 26, IEEE, 2002, pp. 1702–1707, ISBN: 0-7803-7278-6.

[57] A. Narayanan and V. Shmatikov, „How To Break Anonymity of the Netflix Prize Dataset", 2006, ISSN: 1527-974X.

[58] M. Narwaria and S. Arya, „Privacy preserving data mining - 'A state of the art'", *Proceedings of the 10th INDIACom; 2016 3rd International Conference on Computing for Sustainable Global Development, INDIACom 2016*, pp. 2108–2112, 2016.

[59] J. Newsome, B. Karp, and D. Song, „Polygraph: Automatically Generating Signatures for Polymorphic Worms", *2005 IEEE Symposium on Security and Privacy (S&amp;P'05)*, pp. 226–241, ISSN: 1081-6011.

[60] J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra, „A New Approach to Practical Active-Secure Two-Party Computation", pp. 681–700, 2012.

[61] D. Papamartzivanos, F. Gómez Mármol, and G. Kambourakis, „Dendron: Genetic trees driven rule induction for network intrusion detection systems", *Future Generation Computer Systems*, vol. 79, pp. 558–574, 2018, ISSN: 0167739X.

[62] P. Pullonen, „Actively Secure Two-Party Computation: Efficient Beaver Triple Generation", no. august 2013, 2013.

[63] Y. Qi and M. J. Atallah, „Efficient Privacy-Preserving k -Nearest Neighbor Search ",

[64] Z. Qin, J. Weng, Y. Cui, and K. Ren, „Privacy-Preserving Image Processing in the Cloud", *IEEE Cloud Computing*, no. April, pp. 48–57, 2018, ISSN: 23256095.

[65] M. O. Rabin, „How To Exchange Secrets with Oblivious Transfer.", *Technical Report TR-81, Aiken Computation Lab, Harvard University*, pp. 1–5, 1981, ISSN: 0021-1753.

[66] O. Regev, „On Lattices, Learning with Errors, Random Linear Codes, and Cryptography", in *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, ser. STOC '05, New York, NY, USA: ACM, 2005, pp. 84–93, ISBN: 1-58113-960-8.

[67] J. Ristioja, „An analysis framework for an imperative privacy-preserving programming language", no. June, 2010.

[68] R. L. Rivest, L. Adleman, and M. L. Dertouzos, „On Data Banks and Privacy Homomorphisms", *Foundations of Secure Computation, Academia Press*, pp. 169–179, 1978.

[69] R. L. Rivest, A. Shamir, and L. Adleman, „A Method for Obtaining Digital Signatures and Public-key Cryptosystems", *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978, ISSN: 0001-0782.

[70] D. Rossi, M. Mellia, and M. Meo, „Understanding Skype signaling", *Computer Networks*, vol. 53, no. 2, pp. 130–140, 2009, ISSN: 13891286.

[71] G. P. Rout and S. N. Mohanty, „A hybrid approach for network intrusion detection", in *Proceedings - 2015 5th International Conference on Communication Systems and Network Technologies, CSNT 2015*, 2015, pp. 614–617, ISBN: 9781479917976.

[72] Royce Robbins, „Distributed Intrusion Detection Systems: An Introduction and Review", Tech. Rep., 2004, p. 18.

[73] J. E. Rubio, C. Alcaraz, R. Roman, and J. Lopez, „Analysis of Intrusion Detection Systems in Industrial Ecosystems", *Proceedings of the 14th International Joint Conference on e-Business and Telecommunications*, no. Secrypt, pp. 116–128, 2017, ISSN: 1542-6270 (Electronic).

[74] M. Sabhnani and G. Serpen, „Application of Machine Learning Algorithms to KDD Intrusion Detection Dataset within Misuse Detection Context", in *MLMTA*, 2003.

[75] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou, „Specification-based anomaly detection", *Proceedings of the 9th ACM conference on Computer and communications security - CCS '02*, p. 265, 2002, ISSN: 00220167.

[76] A. Shamir and A. Shamir, „How To Share a Secret", *Communications of the ACM (CACM)*, vol. 22, no. 1, pp. 612–613, 1979, ISSN: 14352443.

[77] M. Shaneck, Y. Kim, and V. Kumar, „Privacy preserving nearest neighbor search", *Machine Learning in Cyber Trust: Security, Privacy, and Reliability*, pp. 247–276, 2009, ISSN: 15504786.

[78] B. Shankar, S. Kannan, and C. P. Rangan, „Alternative protocols for generalized oblivious transfer", *Proceedings of the 9th international conference on Distributed computing and networking*, pp. 304–309, 2008, ISSN: 03029743.

[79] R. Shanmugavadivu and D. N. Nagarajan, „Network intrusion detection system using fuzzy logic", *Indian Journal of Computer Science and Engineering*, vol. 2, no. 1, pp. 101–111, 2011.

[80] S. R. Snapp, J. Brentano, G. V. Dias, T. L. Goan, L. T. Heberlein, C.-L. Ho, K. N. Levitt, B. Mukherjee, S. E. Smaha, T. Grance, D. M. Teal, and D. Mansur, „DIDS (Distributed Intrusion Detection System) Motivation, Architecture, and An Early Prototype", *Proceedings of the 14th National Computer Security Conference*, 1991.

[81] S. Soheily-Khah, P. F. Marteau, and N. Bechet, „Intrusion detection in network systems through hybrid supervised and unsupervised machine learning process: A case study on the iscx dataset", *Proceedings - 2018 1st International Conference on Data Intelligence and Security, ICDIS 2018*, pp. 219–226, 2018.

[82] S. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. Chan, „Cost-based modeling for fraud and intrusion detection: results from the jam project, in: DARPA Information Survivability Conference and Exposition", *DISCEX'00, Proceedings*, vol. 2, no. IEEE, 2000, pp. 130–144, 2000.

[83] I. Sumaiya Thaseen and C. Aswani Kumar, „Intrusion detection model using fusion of chi-square feature selection and multi class SVM", *Journal of King Saud University - Computer and Information Sciences*, vol. 29, no. 4, pp. 462–472, 2017, ISSN: 22131248.

[84] T. Tassa, „Generalized oblivious transfer by secret sharing", *Designs, Codes, and Cryptography*, vol. 58, no. 1, pp. 11–21, 2011, ISSN: 09251022.

[85] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, „A Detailed Analysis of the KDD CUP 99 Data Set", in *Proceedings of the Second IEEE International Conference on Computational Intelligence for Security and Defense Applications*, ser. CISDA'09, Piscataway, NJ, USA: IEEE Press, 2009, pp. 53–58, ISBN: 978-1-4244-3763-4.

[86] C. F. Tsai, Y. F. Hsu, C. Y. Lin, and W. Y. Lin, „Intrusion detection by machine learning: A review", *Expert Systems with Applications*, vol. 36, no. 10, pp. 11 994–12 000, 2009, ISSN: 09574174.

[87] C. F. Tsai and C. Y. Lin, „A triangle area based nearest neighbors approach to intrusion detection", *Pattern Recognition*, vol. 43, no. 1, pp. 222–229, 2010, ISSN: 00313203.

[88] F. S. Tsai and K. L. Chan, *Detecting Cyber Security Threats in Weblogs Using Probabilistic Models.* 2007, pp. 46–57, ISBN: 9783642046667.

[89] V. Vapnik and A. Lerner, „Pattern Recognition using Generalized Portrait Method", *Automation and Remote Control*, vol. 24, 1963.

[90] B. Winter, *Computer and Network Security Essentials*, K. Daimi, Ed. Cham: Springer International Publishing, 2018, ISBN: 978-3-319-58423-2.

[91] W. K. Wong, D. W.-l. Cheung, B. Kao, and N. Mamoulis, „Secure kNN computation on encrypted databases", *Proceedings of the 35th SIGMOD international conference on Management of data - SIGMOD '09*, p. 139, 2009, ISSN: 0306-4522.

[92] S. Yakoubov, „A Gentle Introduction to Yao ' s Garbled Circuits Table of Contents",

[93] Y. Yang and J. O. Pedersen, „A Comparative Study on Feature Selection in Text Categorization", in *Proceedings of the Fourteenth International Conference on Machine Learning*, ser. ICML '97, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, pp. 412–420, ISBN: 1-55860-486-3.

[94] A. C.-C. Yao, „Protocols for secure computations", *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pp. 160–164, 1982, ISSN: 0272-5428.

[95] A. C.-C. Yao, „How to generate and exchange secrets", *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, no. 1, pp. 162–167, 1986, ISSN: 0272-5428.

[96] B. Yao, F. Li, and X. Xiao, „Secure nearest neighbor revisited", *Proceedings - International Conference on Data Engineering*, pp. 733–744, 2013, ISSN: 10844627.

[97] W. L. Al-Yaseen, Z. A. Othman, and M. Z. A. Nazri, „Multi-level hybrid support vector machine and extreme learning machine based on modified K-means for intrusion detection system", *Expert Systems with Applications*, vol. 67, pp. 296–303, 2017, ISSN: 09574174.

[98] V. Yegneswaran, P. Barford, and S. Jha, „Global Intrusion Detection in the DOMINO Overlay System", pp. 1–7, 2004.

[99] J. Zhan, L. W. Chang, and S. Matwin, „Privacy-preserving support vector machines learning", *Proceedings of the International Conference on Electronic Business (ICEB)*, pp. 477–482, 2005, ISSN: 16830040.

Fiche masterproef

Student: Henri De Plaen

Titel: Privacy-friendly machine learning algorithms for intrusion detection systems

Engelse titel:

UDC: 51-7

Korte inhoud:

In this thesis, we present a set of practically usable machine-learning algorithms for intrusion detection systems using multi-party computation (secret sharing). This allows a user to query an already trained model preserving both the privacy of the query and of the model. Two classical algorithms used for intrusion detection systems are investigated: support vector machines, both linear and non-linear, and nearest neighbors. Although non-linear support vector machines not seem very suited for the problem, linear support vector machines and nearest neighbors methods deliver very satisfying results. To make nearest neighbors feasible, condensed nearest neighbors are applied beforehand exploiting the trade-off between expensive clear pre-processing and a lightweight secret model. Furthermore, PCA and chi-square feature selection methods are investigated to reduce the evaluation costs even more.

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: wiskundige ingenieurstechnieken

Promotor: Prof. Dr. ir. Bart Preneel

Assessoren: Prof. Dr. ir. Karl Meerbergen
Prof. Dr. ir. Sabine Van Huffel

Begeleiders: Dr. Aysajan Abidin
Dr. Abdelrahaman Aly