Titre: Title:	Low-Rate False Alarm Anomaly-Based Intrusion Detection System with One-Class SVM
Auteur: Author:	Fatemeh Farnia
Date:	2017
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Farnia, F. (2017). Low-Rate False Alarm Anomaly-Based Intrusion Detection System with One-Class SVM [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/2666/

UNIVERSITÉ DE MONTRÉAL

LOW-RATE FALSE ALARM ANOMALY-BASED INTRUSION DETECTION SYSTEM WITH ONE-CLASS SVM

FATEMEH FARNIA
DÉPARTEMENT DE MATHÉMATIQUES ET DE GÉNIE INDUSTRIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INDUSTRIEL)
JUILLET 2017

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

LOW-RATE FALSE ALARM ANOMALY-BASED INTRUSION DETECTION SYSTEM WITH ONE-CLASS SVM

présenté par : FARNIA Fatemeh
en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de :

M. LANGLOIS Pierre, Ph. D., président
M. BASSETTO Samuel, Doctorat, membre et directeur de recherche
M. BRAULT Jean-Jules, Ph. D., membre

DEDICATION

To my family,
To my best friends...

ACKNOWLEDGEMENTS

First, I am highly thankful for the great guidance and encouragement of Dr. Samuel Bassetto over the last two years. Thank you so much for believing in me. Your support was essential for my success here.

I would like to thank Dr. Pierre Langlois and Dr. Jean-Jules Brault for serving as my jury.

Of course, I would like to thank the Groupe Access company which provide me with the real data. Specially Dr. Alireza Sadighian and Dr. Saeed Sarencheh who helped me for this thesis.

RÉSUMÉ

La détection d'anomalie est une tâche consistant à repérer au sein d'un groupe de modèles ceux qui s'écartent de manière significative du comportement attendu ou souhaité. Ces modèles, non conformes, sont appelés anomalies ou données aberrantes. La détection des anomalies comporte diverses applications, telles que la détection des fraudes, la vidéosurveillance, les soins de santé et la détection des intrusions. Généralement, la détection d'anomalies vise à modéliser une fonction d'aide à la décision de manière à ce qu'elle puisse distinguer l'écart, significatif ou non, entre l'anomalie détectée et le comportement classique attendu. La détection des intrusions est maintenant un sujet d'intérêt pour le domaine de la sécurité informatique. Les intrusions (ou anomalies) sont des activités malveillantes permettant de pénétrer un ou plusieurs systèmes afin d'en retirer des informations confidentielles. Par conséquent, nous pouvons utiliser des méthodes de détection d'anomalie pour détecter les intrusions. Afin de pouvoir distinguer les intrusions des comportements classiques ou attendus (modèles non liés à l'attaque), différents systèmes de détection d'intrusion ont été développés. Ces derniers sont divisés en deux catégories: (1) les systèmes basés sur l'anomalie et (2) les systèmes basés sur la signature. Pour ce qui est des systèmes basés sur la signature, les modèles des anomalies sont connus, alors que pour ce qui est des systèmes de détection basés sur l'anomalie, les modèles d'intrusions peuvent être nouveaux. Autrement dit, les systèmes de détection d'intrusions basés sur l'anomalie peuvent détecter de nouvelles anomalies ou des attaques. Cependant, ces systèmes produisent un taux élevé de fausses alertes. Plus précisément, ces systèmes classent par erreur les modèles de non-attaque comme des anomalies, ce qui entraîne un taux élevé de fausses alarmes. Diminuer ce taux est l'un des principaux défis dans les systèmes de détection d'intrusion basés sur l'anomalie.

Il existe plusieurs techniques pour diminuer le taux de fausses alarmes dans les systèmes de détection basés sur l'anomalie, telles que les méthodes de classification d'une classe. Le taux de faux positif montre le taux d'observations de non-attaque qui sont classées comme aberrantes et le taux de faux négatif représente la fraction des valeurs aberrantes qui sont détectées comme des non-attaques. Un système de détection d'intrusion idéal a un taux de faux positif nul et un taux de faux négatif nul. Nous proposons d'utiliser un algorithme nommé *une machine de vecteur de support de classe unique (SVM à une classe)* pour détecter des anomalies, en diminuant le taux de fausses alarmes (faux positif) avec le même taux de vrai positif.

Le SVM d'une classe est un algorithme de classification d'une classe, à savoir une extension

non supervisée de SVM qui construit un modèle basé sur une classe nommée classe cible. Toute observation qui ne se trouve pas dans cette classe s'appelle un horsain. En pratique, cette technique construit une classe pour les observations de cible ou non et une autre classe pour l'origine, et elle tente de trouver un classificateur approprié pour séparer les deux classes. Cette technique vise à calculer un hyperplan qui maximise la distance entre les points de formation et l'origine. Cet hyperplan (fonction de décision) peut être utilisé pour distinguer les observations non vues en tant que horsains ou non-attaques. Sur la base de la définition de cet algorithme, toute observation qui tombe sur le mauvais côté de cette frontière est un dépassement et les autres observations sont classées comme des non-attaques. De plus, cet algorithme profite d'un paramètre important appelé v, qui peut être défini par l'utilisateur pour déterminer la fraction des valeurs aberrantes et des vecteurs de support.

Pour évaluer la méthode proposée, nous utilisons un jeu de données réel capturé à partir des journaux du serveur contenant des indicateurs de bas niveau, tels que la charge de la CPU, l'utilisation du processeur, etc., pour détecter les activités d'intrusion réelle. Nous comparons trois scénarios différents: une SVM de classe unique, une SVM de classe unique avec une étape de réduction de fausses alarmes et une SVM de classe unique avec une étape de réduction de fausses alarmes qui est formée sans valeurs aberrantes. Nous avons utilisé un métrique appelé F-measure pour comparer ces trois scénarios. La mesure F est le moyen harmonique entre précision et rappel. La précision montre la fraction des attaques vraies sur le nombre total d'observations détectées comme attaques, et le rappel représente la fraction des attaques réelles détectées. Sur la base des résultats expérimentaux, la SVM d'une classe avec le scénario de réduction de la fausse alarme a atteint une valeur élevée de F-$0,963$. Plus précisément, la valeur de la mesure F a augmenté de $34,4$ par rapport à la SVM seule et de $52,2$ par rapport à la SVM d'une seule classe avec la réduction de la fausse alarme qui est formée sans valeurs aberrantes. Nous observons que le deuxième scénario a permis de réduire le nombre de fausses alarmes.

Nous comparons également nos scénarios avec trois algorithmes de classification d'une classe: densité de Parzen, mélange d'estimation de densité de Gaussiens et description de données k-means. Les résultats de la description des données k-means et du mélange de Gaussians sont comparables à ceux de la SVM d'une classe avec l'étape de réduction des fausses alarmes, alors que la méthode de densité de Parzen présentait une faible valeur de mesure F de $0,341$. En outre, nous avons testé ces trois scénarios sur un ensemble de données réel (Knowledge Discovery Data Mining 99), sous-tendant encore l'efficacité de la méthode de réduction des fausses alarmes proposée.

ABSTRACT

Anomaly detection is a task of detecting patterns that significantly deviate from an expected behavior. These nonconforming patterns are referred to as anomalies or outliers. Anomaly detection has various applications, such as fraud detection, video surveillance, health care and intrusion detection. Generally, anomaly detection aims to find a decision function such that it can distinguish deviation from the expected behavior. The detection of intrusions is now a topic of great interest in the computer security field. Intrusions (or anomalies) are malicious activities and can penetrate systems and obtain confidential information. Consequently, we can use anomaly detection methods to detect intrusions. For this reason, intrusion detection systems have been introduced to distinguish intrusions from expected behavior (non-attack patterns). Intrusion detection systems are divided into two categories: anomaly-based and signature-based systems. In signature-based systems, the patterns of the anomalies are known, whereas in anomaly-based detection systems, these patterns can be novel. Anomaly-based intrusion detection systems can detect novel anomalies or attacks, although these systems produce a high false alarm rate. Specifically, these systems mistakenly classify non-attack patterns as anomalies, which leads to a high false alarm rate. Decreasing this rate is one of the main challenges in anomaly-based intrusion detection systems.

There are multiple techniques to decrease the false alarm rate in anomaly-based detection systems, such as one-class classification methods. The false positive rate shows the rate of non-attack observations that are classified as outliers, and the false negative rate depicts the fraction of outliers that are detected as non-attacks. An ideal intrusion detection system has zero false positive rate and zero false negative rate. We propose to use an algorithm named *one-class support vector machine (one-class SVM)* to detect anomalies, decreasing the false alarm (false positive) rate with the same false negative rate.

One-class SVM is a one-class classification algorithm, namely, an unsupervised extension of SVMs that constructs a model based on one class named the target class. Any observation that is not in this class is called an outlier. In practice, this technique builds one class for target or non-attack observations and another class for the origin, and it attempts to find a proper one-class classifier to separate the two classes. This technique aims to compute a hyperplane that maximizes the distance between the training points and the origin. This hyperplane (decision function) can be used for distinguishing the unseen observations as outliers or non-attacks. Based on the definition of this algorithm, any observation that falls on the wrong side of this frontier is an outlier, and the other observations are classified as

non-attacks. Moreover, this algorithm takes advantage of an important parameter called v, which can be defined by the user to determine the fraction of outliers and support vectors.

To evaluate the proposed method, we use a real dataset captured from server logs containing low-level indicators, such as CPU load, CPU usage, and so forth, to detect real intrusion activities. We compare three different scenarios: a one-class SVM alone, a one-class SVM with the false alarm reduction step, and a one-class SVM with the false alarm reduction step that is trained without outliers. We use a metric called F-measure to compare these three scenarios. F-measure is the harmonic mean between precision and recall. Precision shows the fraction of true attacks on the total number of observations that are detected as attacks, and recall depicts the fraction of detected real attacks. Based on the experimental results, one-class SVM with the false alarm reduction step scenario achieved a high F-measure value of 0.963. Specifically, the F-measure value increased 34.4 percent compared to the one-class SVM alone and 52.2 percent compared to the one-class SVM with the false alarm reduction step that is trained without outliers. We observe that the second scenario achieved a lower number of false alarms.

We also compare our scenarios with three one-class classification algorithms: Parzen density estimation, mixture of Gaussians density estimation and k-means data description. The results of the one-class SVM with the false alarm reduction step is comparable to those of the k-means data description, whereas the Parzen density method presented the lowest F-measure value of 0.341. Furthermore, we test these three scenarios on one real dataset (Knowledge Discovery Data Mining 99), again underlying the efficiency of the proposed false alarm reduction method.

Finally, we investigate the sample size and dimensionality behavior in one-class SVM with respect to F-measure, training time and FP reduction time on six artificial normally distributed datasets. According to the experimental results, for the same sample sizes, when the dimensionality increases, the F-measure value decreases. Moreover, by increasing the sample size and dimensionality, the training time and FP reduction time also increase.

TABLE OF CONTENTS

DEDICATION iii

ACKNOWLEDGEMENTS iv

RÉSUMÉ v

ABSTRACT vii

TABLE OF CONTENTS ix

LIST OF TABLES xii

LIST OF FIGURES xiii

LIST OF SYMBOLS AND ABBREVIATIONS xv

LIST OF APPENDICES xvi

CHAPTER 1 INTRODUCTION 1
- 1.1 Research methodology 2
- 1.2 Performance analysis 2
- 1.3 Data 5
- 1.4 One-class classification 6
- 1.5 One-class vs two-class classification 7
- 1.6 What is an anomaly? 8
- 1.7 Proposed solution 8
- 1.8 Formal definition of terms 9
- 1.9 Objective of the research 9
- 1.10 Thesis outline 9

CHAPTER 2 CRITICAL LITERATURE REVIEW 11
- 2.1 State of the art in one-class classification 11
- 2.2 State of the art of false alarm reduction 14
- 2.3 Intrusion Detection Systems 16
  - 2.3.1 Challenges of anomaly detection 16
  - 2.3.2 Types of Anomalies 17

2.3.3 Output of Anomaly Detection . . . 18
2.4 Support Vector Machine . . . 18
2.4.1 Kernels . . . 20
2.4.2 Kernel example . . . 23
2.5 One-class SVM . . . 24
2.5.1 Algorithm . . . 26
2.5.2 Optimization . . . 29
2.5.3 Parameters . . . 32

CHAPTER 3 FALSE ALARM REDUCTION METHOD . . . 38
3.1 Proposed false alarm reduction method . . . 38
3.2 Adjusting Parameter T . . . 39
3.3 Toy Example . . . 41
3.4 Comparison with other methods . . . 45

CHAPTER 4 TEST . . . 46
4.1 Real Dataset . . . 46
4.2 Data Preprocessing . . . 47
4.3 Model Selection . . . 51
4.4 Visualization Methods . . . 52
4.4.1 ROC Graph . . . 52
4.4.2 Confusion Matrix Plot . . . 53
4.5 Empirical Result . . . 54
4.5.1 Scenario 1 . . . 54
4.5.2 Scenario 2 . . . 56
4.5.3 Scenario 3 . . . 58
4.5.4 Comparison of the three scenarios . . . 62
4.6 Empirical results on other algorithms . . . 66
4.7 Empirical result on KDD99 dataset . . . 67
4.8 Impact of the sample size . . . 68
4.9 Discussion of empirical result . . . 69

CHAPTER 5 CONCLUSION . . . 70
5.1 Advancement of knowledge . . . 70
5.2 Limits and constraints . . . 71
5.3 Recommendations . . . 72

REFERENCES 73

APPENDICES 77

LIST OF TABLES

Table 1.1 Confusion matrix example 4
Table 2.1 Experimental results for different values of parameter ν 36
Table 2.2 Experimental results for different values of parameter γ 36
Table 3.1 Euclidean distances on toy example 44
Table 3.2 Performance of one-class SVM on toy example 45
Table 4.1 Average performance of one-class SVM in 10 different runs 52
Table 4.2 Performance of one-class SVM in three scenarios 63
Table 4.3 Comparison of one-class classification methods 67
Table 4.4 Experimental results of two scenarios on the KDD99 dataset 68
Table 4.5 Experimental results for various sample sizes 68
Table A.1 Observations 77
Table A.2 Kernel matrix 78
Table A.3 Optimization table 81
Table B.1 Selected parameters of one-class SVM in 10 different runs (robust scaling) . 84
Table C.1 Selected parameters of one-class SVM in 10 different runs (Lasso) 85

LIST OF FIGURES

Figure 1.1	The ROC curve	5
Figure 1.2	One example of a one-class classifier	7
Figure 2.1	Gaussian model	12
Figure 2.2	One example of contextual anomaly	17
Figure 2.3	2D toy example of binary classification(Possible separating hyperplanes)	19
Figure 2.4	2D toy example of binary classification (Maximum Margin classifier)	19
Figure 2.5	2D toy example of binary classification (Soft Margin)	20
Figure 2.6	2D toy example of binary classification(Non-linear classifier)	21
Figure 2.7	3D toy example of binary classification(Non-linear classifier in input space but linear classifier in feature space)	21
Figure 2.8	One-class SVM hyperplane without the introduction of kernels (left), and one-class SVM hyperplane with the introduction of kernels (right)	26
Figure 2.9	A 2D toy example showing one-class SVM hyperplane	28
Figure 2.10	Scatter plot of spherical 2D toy example	32
Figure 2.11	Spherical 2D toy example with $\nu = 0.05$ and four different γ values	33
Figure 2.12	Spherical 2D toy example with $\nu = 0.2$ and four different γ values	34
Figure 2.13	Spherical 2D toy example with $\nu = 0.5$ and four different γ values	35
Figure 2.14	Scatter plot of banana-shaped 2D toy example with polynomial kernel	37
Figure 3.1	ROC score versus parameter T	42
Figure 3.2	Scatterplot	42
Figure 3.3	Scatterplot with decision boundary with one-class SVM	43
Figure 3.4	Scatterplot with decision boundary after reducing false alarms	45
Figure 4.1	The histogram plot of all the features in the train file	48
Figure 4.2	Line plot for all the features	49
Figure 4.3	Basic ROC graph showing two classifiers	53
Figure 4.4	Basic confusion matrix plot	54
Figure 4.5	Project structure of one-class SVM (scenario 1)	55
Figure 4.6	Scatterplot with decision boundary (scenario 1)	56
Figure 4.7	AUC versus parameter T (scenario 2)	58
Figure 4.8	Project structure of one-class SVM (scenario 2)	58
Figure 4.9	Scatterplot with decision boundary (scenario 2)	59
Figure 4.10	Project structure of the one-class SVM (scenario 3)	60
Figure 4.11	AUC versus parameter T (scenario 3)	61

Figure 4.12 Scatterplot with decision boundary (scenario 3) 62
Figure 4.13 Confusion matrix plot for one-class SVM using the data without outliers (scenario 1) 64
Figure 4.14 Confusion matrix plot for one-class SVM using the data without outliers (scenario 2) 64
Figure 4.15 Confusion matrix plot for one-class SVM using the data without outliers (scenario 3) 65
Figure 4.16 ROC graph for one-class SVM (all three scenarios) 65
Figure 4.17 ROC curves for various one-class classification algorithms and three scenarios 67

LIST OF SYMBOLS AND ABBREVIATIONS

AUC	Area Under the ROC Curve
BN	Bayesian Network
FN	False Negative
FP	False Positive
KDD99	Knowledge Discovery Data mining 99
KKT	Karush-Kuhn-Tucker
NN	Neural Network
PCA	Principle Component Analysis
RBF	Radial Basis Function
ROC	Receiver Operating Characteristics
SOM	Self Organizing Map
SMO	Sequential Minimal Optimization
SRM	Structural Risk Minimization
SV	Support Vectors
SVDD	Support Vector Data Description
SVM	Support Vector Machine
TN	True Negative
TP	True Positive

LIST OF APPENDICES

ANNEX A NUMERICAL EXAMPLE 77

ANNEX B EMPIRICAL RESULT OF ROBUST SCALING 83

ANNEX C EMPIRICAL RESULT OF LASSO REGRESSION AS FEATURE SELECTION METHOD 84

CHAPTER 1 INTRODUCTION

IBM recently conducted research on the cost of data breaches of 383 companies (Ponemon and IBM, 2016), finding that the average cost of one breach increased from 3.79 to 4 million dollars between 2015 and 2016. This cost increase is due to the change in the average cost for each stolen record in these companies. In fact, each stolen record cost has increased from $154 to $158 between these years. Data breaches occur when there is an attack on or an intrusion into the systems, and they produce extra work for the network. Researchers introduced the idea of intrusion detection systems with the ability to distinguish attacks from non-attack observations to secure systems from intruders. Hence, detection systems tend to minimize the increase of extra work as well as detecting deviations and anomaly patterns.

One main type of intrusion detection system is anomaly based, which aims to distinguish novel anomalies from non-attack patterns. However, current anomaly-based intrusion detection systems suffer from a high false alarm rate. According to a report by Damballa & Ponemon, who surveyed 630 IT departments, the average annual cost of addressing alarms is 1.27 million dollars (Ponemon and Damballa, 2015). In IT departments, experts spend time evaluating all the alarms detected by intrusion detection systems, although most of these alarms are not true intrusions. Clearly, by decreasing the false alarm rate, we will consequently decrease the undesired costs of addressing false alarms. Therefore, we propose a method to be used as an effective way to decrease this rate in the provided data.

The general question under consideration in this thesis is how to construct a single anomaly-based intrusion detection system that can detect novel anomalous behavior and decrease the rate of false alarms. Specifically, is it possible to use detected outliers for decreasing this rate.

1.1 Research methodology

Anomaly-based intrusion detection systems can detect novel attacks, but they produce a high false alarm rate. Considering this challenge of current anomaly-based intrusion detection systems, we investigate the idea of reducing false alarms in this research. In the literature review, it is possible to observe that the research in this domain is ongoing. For instance, Landress (2016) recently utilized a combination of machine learning techniques to decrease the false alarm rate. In addition, Xiao and Li (2008) took advantage of an outlier detection algorithm for this purpose. However, in most cases, the anomaly detection problem has been addressed using algorithms that are designed for one-class classification problems, such as one-class SVM, *support vector data description (SVDD)* and k-means. We select one-class SVM for detecting anomalies, and we also used this algorithm for the problem of high false alarm rates. In greater detail, we propose a false alarm reduction step based on outliers and support vectors (SVs) detected by this algorithm.

We compare our model with two scenarios to show that the proposed method can achieve better performance than these scenarios. Specifically, the proposed method is compared to a one-class SVM without the false alarm reduction step and to a one-class SVM trained based on training points without outliers. Moreover, we investigate the accuracy of other one-class classification methods, such as Parzen density estimation, mixture of Gaussians and k-means data description, on the given data. All the details regarding the indicators to test our proposed method are provided in the next section.

We also test our model on one toy example described in chapter 3 and on the KDD99 dataset explained in chapter 4.

1.2 Performance analysis

This section starts by introducing eight evaluation metrics: true positive, false positive, false negative, true negative, precision, recall, *receiver operating characteristic (ROC)* score, and F-measure. Next, we provide one example to better illustrate these metrics.

Terms including *true positive (TP)*, *true negative (TN)*, *false positive (FP)* and *false negative (FN)*, as well as their combinations, are popular approaches for denoting the performance of classification techniques. If a true and predicted class of the observation is positive, then it is called a TP. If a negative instance is classified as positive, then it is named a FP. If a negative observation is classified as negative, then it is counted as a TN. Finally, if a positive value is classified as negative, then it is called a FN.

In the anomaly detection domain, FP shows the false alarm rate, and FN shows the attacks that are not detected by the anomaly detection system. Consequently, TP shows the rate of detecting attacks, and TN shows the rate of accepted non-attack observations (Mokarian et al., 2013).

Recall, also known as *sensitivity* or *TP rate*, describes the detected percentage of positive instances, as shown in equation 1.1. If the recall value is equal to one, it means that the algorithm detected all the positive, or attack, instances (Ting, 2011).

$$recall = \frac{TP}{TP + FN} \tag{1.1}$$

Precision depicts how successful the algorithm is in detecting real positive observations, as shown in equation 1.2. Specifically, by using precision, it is possible to calculate the extent to which the positive values are actually TPs. A lower FP rate clearly corresponds to a higher precision value (Ting, 2011).

$$precision = \frac{TP}{TP + FP} \tag{1.2}$$

F-measure is also a method for model evaluation that calculates the weighted harmonic mean of recall and precision, as presented in equation 1.3. In greater detail, F-measure is a compromise between precision and recall. If the value is close to one, it indicates that the classifier is proper to use, whereas if the F-measure value is close to zero, it means that the classifier has failed in detecting the outliers, detecting non-attack observations or both. In addition, the value of F-measure is high whenever precision and recall are high. In other words, if the rate of FN or FP decreases, then the F-measure rate increases.

$$F - measure = \frac{2}{1/precision + 1/recall} \tag{1.3}$$

ROC score or *area under the ROC curve (AUC)* is a scalar value that shows the performance of a classifier (Fawcett, 2006). In fact, this value shows the area under the ROC curve. AUC is a portion of a unit square that has a value between 0 and 1, and it is possible to compute the AUC using the following equation:

$$AUC = \frac{x_2 * y_2}{2} + \frac{(1 - x_2)(1 + y_2)}{2} \tag{1.4}$$

,

where x_2, y_2 are equal to $(FPrate, TPrate)$. Note that this equation is for discrete classifiers where we only have one value of FP rate versus one value of TP rate.

The ROC curve is a 2D plot that shows the TP rate on the Y axis versus the FP rate on the X axis, and they are plotted in a unit square called ROC space (cf. Figure 1.1). TP rate is the fraction of TPs among the total number of positive examples, and FP rate is the fraction of FPs among the total number of negative examples. This curve is used to show the performance of a binary or one-class classifier. One example of this graph is presented at the end of this section; however, more details about the ROC graph are discussed in section 4.4.1.

Now, suppose that we trained a one-class classifier and that we calculated its confusion matrix, as shown in Table 1.1. In the confusion matrix, the non-diagonal values of the matrix depict the misclassified observations, and the diagonal values represent the number of instances that are correctly classified.

Table 1.1 Confusion matrix example

Assigned label	True class	
	Attack	Non-attack
Attack	8	2
Non-attack	4	20

According to table 1.1, we can compute recall (TP rate), precision, F-measure and FP rate as follows:

$recall = \frac{TP}{TP+FN} = \frac{8}{8+2} = 0.8$

$precision = \frac{TP}{TP+FP} = \frac{8}{8+4} = 0.667$

$F - measure = \frac{2}{1/precision+1/recall} = \frac{2}{1.25+1.499} = 0.728$

$FP\ rate = \frac{FP}{N} = \frac{4}{24} = 0.167$

Figure 1.1 shows the ROC curve for this classifier. The pair (FP rate, TP rate) are illustrated by point (0.167,0.8), and we can plot the ROC curve. Moreover, the yellow part depicts the area under this ROC curve, and we can use the point $x_2, y_2 = (0.167, 0.8)$ to calculate one single value for this curve. Thus, we can compare different classifiers based on this single value, i.e., based on the AUC. Note that this classifier is discrete; thus, we only have three points for each curve. Based on these values, we can compute the AUC value for this numerical example as follows:

$AUC = \frac{x_2 * y_2}{2} + \frac{(1-x_2)(1+y_2)}{2} = \frac{(0.167*0.8)}{2} + \frac{0.833*1.8}{2} = 0.0668 + 0.749 = 0.816$

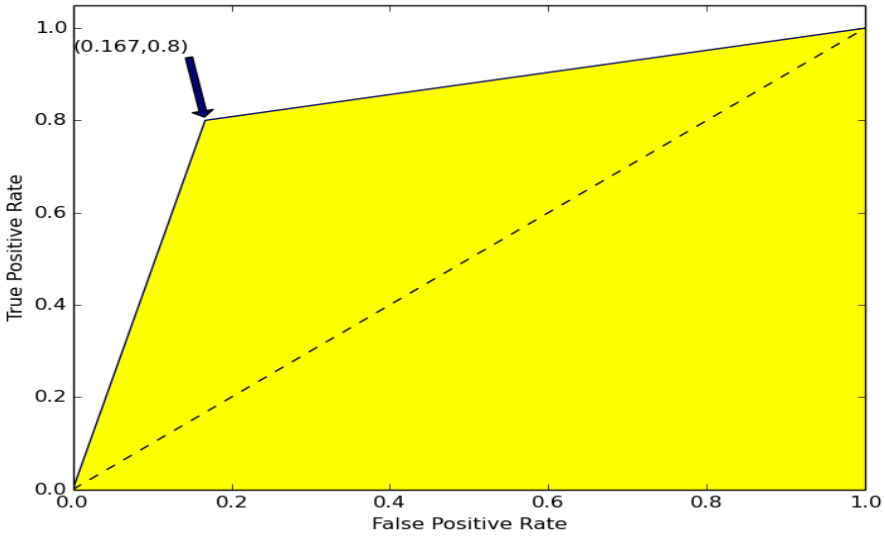

Figure 1.1 The ROC curve

1.3 Data

In this thesis, the *inputs*, *observations*, *patterns*, *examples* and *instances* are described as vectors of d dimensions, $\mathbf{x}_i = (x_{i,1}, ..., x_{i,d})$, where $\mathbf{x}_i \epsilon \mathcal{X}$ is a non-empty set and $\mathcal{X} \epsilon \mathbf{R}^N$. Here, the boldface letters indicate a one d-dimensional vector. Each of these inputs is assigned a label shown by $y_i \epsilon \{-1, 1\}$ as *outputs*. Moreover, the number of components of each vector depicts the number of *features* since each observation has some characteristics.

The provided real data consist of rows depicting examples and columns showing features. These features are low-level indicators captured from server log files, such as CPU usage, CPU load and so forth. These data are in two files: one containing non-attack observations and one capturing non-attack and attack observations. These two datasets are provided by the *Groupe Access* company, and they were captured from server log files. We have started a project with the title of "A machine learning method for anomaly-based intrusion detection system". We aimed to construct a system that can detect attacks with the contribution of reducing the false alarm rate. Thus, the main perspective of this research is on reducing this rate based on one-class SVM algorithm. *Groupe Access* was founded in 1993, and it provides information technology and hardware services, such as data protection and recovery, infrastructure core network design and management, and monitoring and customizing private, public, and hybrid clouds.

We combined these two datasets into a single dataset, leading to a dataset with a large

sample size (number of observations) of non-attack observations and a small sample size of attack observations. Whenever there are no examples of one of the classes or there are few examples of them, the use of one-class classification methods is suggested (Tax, 2001). Hence, our problem is a one-class classification, and we want to find a one-class classifier that can separate these non-attack observations from attack patterns with minimum error. A *one-class classifier* or *frontier* is a decision function that decides the label of each observation. Note that a good one-class classifier is one with a small fraction in both FN and FP rates.

Note that in the domain of anomaly detection, a label of 1 indicates attack observations and a label of -1 depicts non-attack instances. Although the meanings of the labels are opposite in one-class classification, we can still use this algorithm for the anomaly detection problem. However, we have to reverse all the class labels to be consistent with the anomaly detection domain and to have a meaningful value for the performance metrics presented in section 1.2.

1.4 One-class classification

In binary classification, we have examples of two classes, and we want to separate them from each other using a classifier. However, in one-class classification, only the instances of *one* class are well sampled (Tax, 2001). In fact, the observations of another class called *outlier* are either absent or are not a good representative of its underlying distribution. Consequently, we need to find a description for this one class called the *target* class. Moreover, the classifier should be established such that only the target instances are accepted, and all the test observations belonging to the outlier class should be rejected. In greater detail, in solving one-class classification problems, it is assumed that the target class is sampled well and that the other class is not sampled well or does not exist. For example, in the fault detection domain, the information about the system being in a normal state is easily provided, whereas creating the faulty situations are quite costly or even impossible. In these types of examples, the target class is well sampled, but the outlier class could be anything.

For solving one-class classification problems, we want to minimize two errors, i.e., FN and FP. We define the number of FNs as type I error ε_I and the number of FPs as type II error ε_{II} (we use these terms interchangeably). Thus, the classifier should be calculated such that it minimizes both ε_I and ε_{II}. Note that if there is no example of the outlier class, then minimizing the FN rate leads to a classifier that labels all the observations in the training set as targets. Therefore, we need to use patterns of the non-target class to find the best trade-off between minimizing these two errors. Non-target observations are the patterns that we know should be rejected by the one-class classifier. The existence of these patterns can be used to find a tighter boundary that can generalize better. Consequently, these observations

should fall outside the one-class classifier known as the outlier class (Tax, 2001).

1.5 One-class vs two-class classification

In two-class or binary classification, the classifier is supported from both sides, whereas in one-class classification, the classifier is supported only from one side. Additionally, in two-class classification, the objective is to find a smooth classifier. However, in one-class classification, not only do we look for a smooth classifier but we also look for a closed boundary around the data, i.e., a boundary in all directions of the data. Thus, the problem of one-class classification is more difficult than two-class classification, and this problem requires more data points to specify the boundary around the data (Tax, 2001; Khan and Madden, 2014).

The challenges of the conventional classification problems, such as estimating the classification error, the curse of dimensionality, measuring the complexity of the classifier, and generalizing the calculated model, become more prominent in one-class classification problems.

One example of a one-class classifier is presented in figure 1.2. The red line shows one possible binary classifier, and the green circle is one possible one-class classifier. Any point that falls inside this green line is a target pattern, and any point that falls outside this green line is considered to be an outlier or non-target pattern.

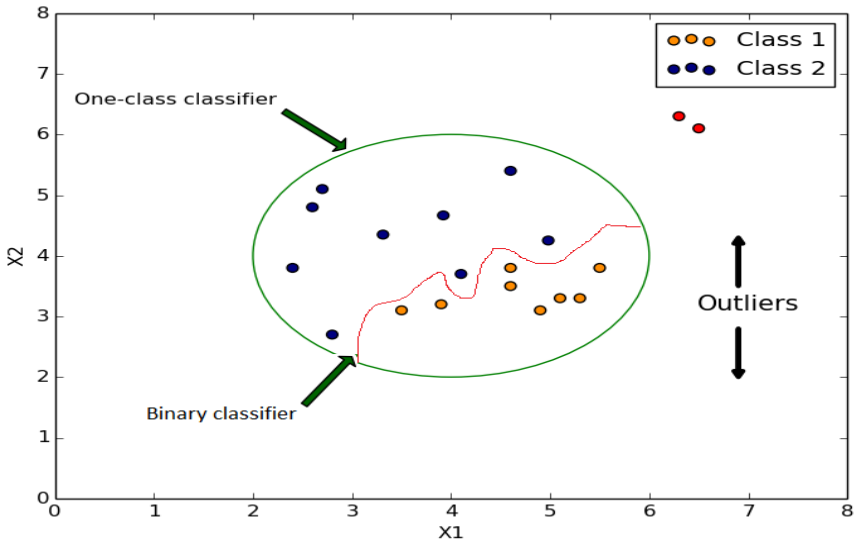

Figure 1.2 One example of a one-class classifier

1.6 What is an anomaly?

Anomalies or outliers are observations in the data that do not conform with the expected behavior. Figure 1.2 shows a simple example of anomalies in red points in 2D dataset. In fact, there is one safe area (green circle) in this figure and any point that falls outside this safe area is considered as anomaly. In this figure, we can visually detect the outliers since they are not located in this safe area. However, there are cases where it is not visually possible to detect and we need machines to detect them. Besides, in general, outliers have low probability of occurrence.

1.7 Proposed solution

There are multiple ways to construct anomaly-based intrusion detection systems, such as using machine learning techniques and data-mining-based approaches. The choice of a proper algorithm for these types of problems highly depends on the data. As explained in section 1.3, we have a small sample size (number of training observations) of attack examples and a large sample size of non-attack patterns to train our model; thus, we face a one-class classification problem. Several methods are available to solve these types of problems, such as density estimation, boundary estimation and reconstruction techniques, which are discussed in greater detail in section 2.1.

In this thesis, we use a boundary algorithm known as one-class SVM to detect anomalies and simultaneously decrease the false alarm rate. One-class SVM is an algorithm for estimating a function that can be used to detect attack patterns. Moreover, in this algorithm, the fraction of detected anomalies or outliers can be defined by the user. There is ongoing, state-of-the-art research using this algorithm in anomaly-based intrusion detection, either by combining this technique with other machine learning techniques such as deep learning (M.Erfani et al., 2016) or by enhancing the algorithm itself (Yin et al., 2014). These articles suggest that this is a promising method for detecting anomalies. Based on the advantages of this algorithm and the provided data characteristics, we propose using this algorithm to detect anomalies and also using these anomalies and SVs to reduce the FP rate.

Again, as explained in section 1.3, only a few attack examples are provided for training. Therefore, this is a one-class classification problem, and we can solve these types of problems with algorithms such as one-class SVM.

1.8 Formal definition of terms

As Schölkopf and Smola explained in their book (Schölkopf and Smola, 2002), outliers are data instances that fall on the wrong side of the hyperplane. Note that when using one-class classification algorithms such as one-class SVM in the anomaly detection domain, the terms *anomalies* and *attacks* are the same as outliers. Thus, we can conclude that all of these terms are the observations that lie on the wrong side of the hyperplane. Moreover, if we have some observations with the attack label, we can use them in model selection. Attack examples should be rejected by the trained one-class classifier (Tax, 2001). Moreover, *non-attack* observations are the patterns that fall inside the decision boundary. Therefore, non-attack observations are members of the target class, and attack observations are members of the outlier class. One-class SVM can calculate a hyperplane named *decision function*, *one-class classifier* or *frontier* that can separate target instances from outlier patterns. Hence, if one observation falls on the correct side of the classifier, then it is classified as a non-attack or target, and if it is rejected by the frontier, then it is classified as an attack or non-target.

1.9 Objective of the research

This research has one main objective: reducing the false alarm rate in anomaly-based intrusion detection systems. There is a contribution of using SVs and outliers based on the one-class SVM algorithm to decrease this rate.

1.10 Thesis outline

This thesis is organized as follows:

- In the second chapter, the state of the art in one-class classification and false alarm reduction are introduced. Moreover, brief explanations of intrusion detection systems, SVM and one-class SVM are presented.
- The third chapter discusses the proposed false alarm reduction technique.
- The fourth chapter starts with the data description, data pretreatment and continues with the implementation of one-class SVM in three different scenarios and comparing them. Then, a section is presented that discusses the experimental result based on three one-class classification algorithms. Next, there is a summary of the result of the proposed method on the KDD99 dataset. Finally, the impacts of sample size and dimensionality are discussed.

- The final chapter presents the conclusions, limitations and future work for this research.

CHAPTER 2 CRITICAL LITERATURE REVIEW

This chapter begins with the state of the art in false alarm reduction, followed by a description of intrusion detection systems, SVM and one-class SVM.

2.1 State of the art in one-class classification

In general, one-class classification methods can be categorized based on three important aspects: the availability of the training examples, the algorithms and the domain in which they have been applied.

a) Availability of training examples is divided to two parts: target examples and unlabeled data are available, target examples and some undersampled non-target observations or artificial outliers are available.

Note that in one-class classification problems, labeled data are difficult to capture, whereas unlabeled data are readily available (Khan and Madden, 2014).

One-class SVM and SVDD use both target and outlier examples to train their models. Tax (2001) used artificial outliers to find an enclosed boundary that encompasses all the positive examples as the members of the target class. He proposed an algorithm called SVDD that aims to calculate a closed boundary around a target data set, i.e., a hypersphere. The sphere is shown by its center a and radius R, and it wants to locate all the training patterns located in the sphere. Specifically, it aims to calculate a hypersphere function with minimum volume around the target class. Similar to all algorithms based on SVs, this algorithm uses slack variables to have a more flexible classifier. Hence, we want to minimize the volume of the hypersphere (R^2) and the sum of the distances to the ξ_i from the objects x_i, leading to the following minimization problem:

$$\epsilon(R, a, \xi) = R^2 + C \sum_i \xi_i, \tag{2.1}$$

with the following constraints:

$$\|x_i - a\| \leq R^2 + \xi_i \xi_i \geq 0. \tag{2.2}$$

It is possible to monitor the tradeoff between errors and the volume of the decision boundary through a user-defined parameter C.

b) The one-class classification algorithms are divided to three categories: Density, boundary and Reconstruction methods.

The most straightforward solutions are density algorithms, such as Gaussian models, *mixture of Gaussians (MoG)* and Parzen density, with making the assumption of a Gaussian-like distribution. Since these algorithms compute a complete description of the training points, they require a large number of such points. Therefore, these algorithms are a good choice when the sample size is sufficient and when the assumed distribution is correct. Figure 2.1 presents an example of a Gaussian model, where the yellow areas show the region in which the non-target observations should fall.

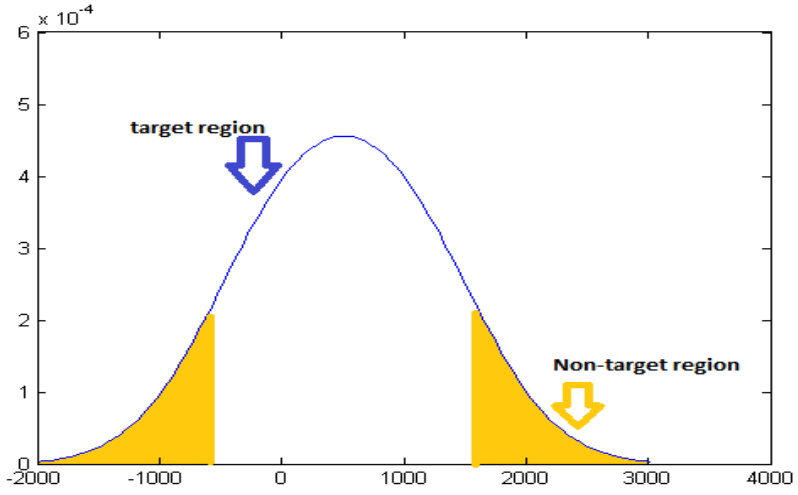

Figure 2.1 Gaussian model

Making the assumption of the Gaussian model requires that the data should be unimodal and convex. This strong imposition leads to strong assumptions, and in most cases, the data do not follow this distribution (Tax, 2001). Consequently, a more flexible version was introduced, known as MoG (Duda et al., 1973). This model requires more training patterns compared to Gaussian models since it is linear combinations of individual Gaussian models, as shown in equation 2.3. In fact, we assume that each feature has a unique Gaussian model. Furthermore, more data are needed for this method compared to a single Gaussian model.

$$p_{MoG}(x) = \frac{1}{N_{MoG}} \sum_j \alpha_j P_N(x; \mu, \sum_j) \tag{2.3}$$

In equation 2.3, N_{MoG} stands for the number of Gaussians, and α_j are the mixing coefficients. μ_j and $\sum_j$ are the mean and covariance for the jth Gaussian. These values can be measured

using the expectation maximization algorithm (Bishop, 1995). *Parzen density estimation* is an extension of MoG, and they are a mixture of Gaussians and other kernels centered on each of the training points (Parzen, 1962). This estimator assumes an equal value for all the coefficients; thus, it has sensitivity to scaling of the data.

$$p_p(X) = \frac{1}{N} \sum_i P_N(x; x_i, hI) \tag{2.4}$$

where h stands for the width of the kernel, and it is optimized through the maximum likelihood. In fact, h is the only parameter that should be optimized; therefore, the computation time for this algorithm is almost zero. Moreover, the covariance matrix is shown by $\sum_i = hI$ in equation 2.4.

In boundary methods, we want to estimate a closed boundary around the data points, i.e., a decision boundary. Since we only look for this boundary, the number of training examples required for training the model is less than that for density methods (Tax, 2001). Note that boundary techniques are sensitive to the scaling of features since they use distances to define the boundary. Algorithms such as k-centers, one-class SVM and SVDD are in this category. k-centers employs k small balls around the training patterns with the same radius value, and the centers of the balls are on the training patterns (Ypma and Duin, 1998). This algorithm aims to optimize the radius and centers to have all these patterns in the ball.

Reconstruction methods use the information provided in the data and then assumes one generation process for it. k-means is a construction method that assumes that the data are clustered and can be demonstrated by some clusters (Bishop, 1995).

c) the application domains of one-class classification algorithms are divided to two categories, anomaly detection and other applications.

Anomaly detection is one of the main applications where the algorithms to solve one-class classifications problems are used.

Li et al. (2003) presented an improved version of the algorithm proposed by Schölkopf et al. (2001) for the purpose of anomaly detection. They proposed considering all the observations close to the origin as outliers. In this case, the origin will not be considered as the only member of the second class as in the Schölkopf et al. (2001) algorithm.

Zhang et al. (2015) adopted a one-class SVM for training an anomaly detection system. They focused on the unavailability of the attack patterns in a network-based intrusion detection system. Moreover, they compared their proposed method with NN and C-SVM, finding that the one-class SVM technique has comparable performance in terms of F-measure, recall and

precision.

Other applications include handwritten detection (Schölkopf et al., 2001), (Tax, 2001); machine fault detection (Shin et al., 2005) ,(hassan et al., 2015); bioinformatics (Yousef et al., 2008); and text classification, among others.

2.2 State of the art of false alarm reduction

False alarm reduction is divided into two categories: approaches that reduce this rate in the detection phase, and alert processing techniques, which identify the patterns of false alarms after the detection phase. In this thesis, we aim to decrease the false alarm rate at the detection phase, so the focus here is on the articles which are related to similar issues.

Xiao and Li (2008) took advantage of frequent pattern-based outlier detection to reduce the false alarm rate. Moreover, they used weights for the data features in order to control the effect of these features on the reduction of false alarms. Precisely, these weights, or scores, demonstrate the abnormality of false alarms. Higher scores were given when the instances have a more frequent pattern. The main advantage of their technique is that it is an online method, and can learn from new false alarms and report the scores to the intrusion detection expert. They tested their method on two different data sets consisting of different types of alerts, and found a reduction in false alarms for two of them.

Om and Kundu (2012) combined the k-means and *k-nearest neighbors* and the naive Bayes classifier in order to build an anomaly detection system and reduce the false alarm rate. The proposed method makes use of a feature selection algorithm to discover the principal features and eliminate the irrelevant ones from the given data. In addition, it can also detect the type of intrusion. The k-means algorithm was employed to separate the data into normal and anomaly clusters, while the K-nearest neighbors algorithm and naive Bayes classifier play the training model role, based on the labeled observations. They suggested that through this combination of clustering and classification, it is possible to reduce the false alarm rate of the intrusion detection system.

Juma et al. (2014) suggested combining X-mean clustering and a random forest classifier to tackle anomaly detection and minimize the false-alarm rate. In the first step, the anomaly and normal observations are divided into two clusters via the X-mean clustering algorithm. Then, these labeled observations are re-classified into the proper classes by the random forest method. Through this combination they achieved a decrease in the false alarm rate, with high detection rate.

Narsingyani and Kale (2015) used a *Genetic Algorithm* for an anomaly-based intrusion detection system in order to reduce the rate of false alarms. First, a set of classification rules is created, and then, based on these rules, a new unseen observation is classified to its proper class. With this technique they achieved a lower false alarm rate using *Knowledge Discovery and Data Mining Cup 99(KDD99)* database. This database consists of normal and attack observations, where the attacks demonstrate multiple types.

Landress (2016) employed several machine learning techniques to decrease the false alarm rate. The method begins with feature selection using decision tree, continues with unsupervised K-means for clustering the data into different categories, and finally reduces false alarms using *Self Organizing Map(SOM)*. SOM maps the feature space to a two-dimensional topological map to find neighbors in the grid. They used K-means in an unsupervised setting, demonstrating a substantial reduction of false alarms with this combination of machine learning techniques. They tested their method on KDD99 dataset. In average, they achieved 96.8 percent accuracy for three clusters.

Li et al. (2016) looked at the problem of false alarms using a multiple-view method, looking at the data in both source and destination feature sets. They introduced a semi-supervised method to solve the anomaly detection problem and simultaneously reduce false alarm rate of network-based intrusion detection systems. Specifically, they divided the data into two views, the destination feature set and source feature set. Comparing their method with several similar methods, they found that their approach results in greatly reducing the false alarm rate.

Goeschel (2016) proposed a false alarm reduction method based on SVM, naive Bayes classifier and decision tree. Their approach is consisted of three steps in order to achieve low false alarm rate in detecting novel attacks. At the first step, SVM makes decision whether an observation is normal or attack, based on the new attribute added to the dataset. At the second step, each attack observation is checked by decision tree algorithm to verify whether they are novel or not. At the third step, the type of the attack is determined through a voting procedure by decision tree and naive Bayes. They proved their hybrid model decreases the false alarm rate (1.57 percent) without any impact on the detection based on the KDD99 dataset.

Some of these aforementioned techniques may have one or more main drawbacks, including human dependency, large computation time and the need for using attack patterns to build the model. Human dependency is when there is a need for human interaction in the detection phase. For instance, Xiao and Li (2008) use little human interactions for their method. large computation time is due to high number of features. The need for using attack patterns to

build the model is a shortage in the case of not knowing their pattern in the training phase.

2.3 Intrusion Detection Systems

Intrusions are malicious activities that invade security systems. Intrusion detection is used to identify intrusions in a system. The main assumption in this context is that the intrusion results an abnormal system behavior (such as higher CPU load, higher memory usage and increase in fan speed). Hence, the intrusion pattern is entirely different from the pattern of non-attack instances. The recent increase in new types of intrusive attacks has necessitated the need to build *intrusion detection systems* based on machine learning methods. Such machine learning methods are able to devote broad attention to detect malicious behavior (Omar et al., 2013), and they are used in many intrusion detection systems.

From the information source's point of view, intrusion detection systems are categorized into two sections, host-based and network-based systems (Devarakonda et al., 2012). Host-based intrusion detection systems gather data from system logs and audit records, whereas network-based intrusion detection systems collect data from internet packets (Anderson, 1980). Another categorization of intrusion detection systems from the detection point of view is *misuse detection* and anomaly detection. Misuse, or signature-based, detection has the ability to detect known intrusions with a low rate of false alarms. This form of intrusion detection systems is used when the types of the attacks are known. However, misuse detection fails to detect novel intrusions (Mukkamala et al., 2002), which can be detected by anomaly detection systems. Anomaly detection has many disadvantages, such as complexity and a high false alarm rate. Anomalies or outliers are patterns that are different from expected behavior. There are numerous reasons for anomalies, such as credit card fraud, cyber-intrusion, breakdown of a system, military surveillance, etc.

In the field of the anomaly detection, several crucial aspects of anomalies have been identified, including the challenges of anomaly detection, types of anomalies, and the output of anomaly detection. These are described in the following sections.

2.3.1 Challenges of anomaly detection

There are three main types of challenges involved with anomaly detection (Chandola et al., 2009). First, anomalies can take various patterns in different domains. Patterns are regularities in the data (Bishop, 2006). For instance, in the medical domain the significant difference between anomalous and normal patterns can be small, whereas in industrial domain they can be extremely large. Second, the anomalies may change over time, which demands the updat-

ing of anomaly-based intrusion detection system in order to achieve consistent performance. Third, in several cases, distinguishing between patterns of normal and abnormal behavior is not possible, since they may have the same overall pattern.

2.3.2 Types of Anomalies

In this section, we only discuss two types of anomalies, as follows (Chandola et al., 2009):

Point anomalies are the simplest and most common type of anomalies found in anomaly detection processes. If an independent point can be identified as an anomaly compared to the normal data points, it is called a point anomaly. In the domain of intrusion detection, we encounter point anomalies. Figure 1.2 shows one example of this type of anomaly.

Contextual anomalies are instances that are anomalous in a specific context but not considered anomalies in other contexts. Each data instance can be defined using *Contextual attribute* and *Behavioral attribute*. A contextual attribute defines the context of the instances, and a behavioral attribute depicts the non-contextual aspects of the instances. This kind of anomaly is frequently found in time-series and spatial data. Figure 2.2 depicts one contextual anomaly in a precipitation time series data. As you can see in time T2 there is an anomaly. A behavior that is not expected at this time, however, this behavior in time T1 is expected. Hence, this behavior is tolerated in T1 and it is an anomaly in time T2.

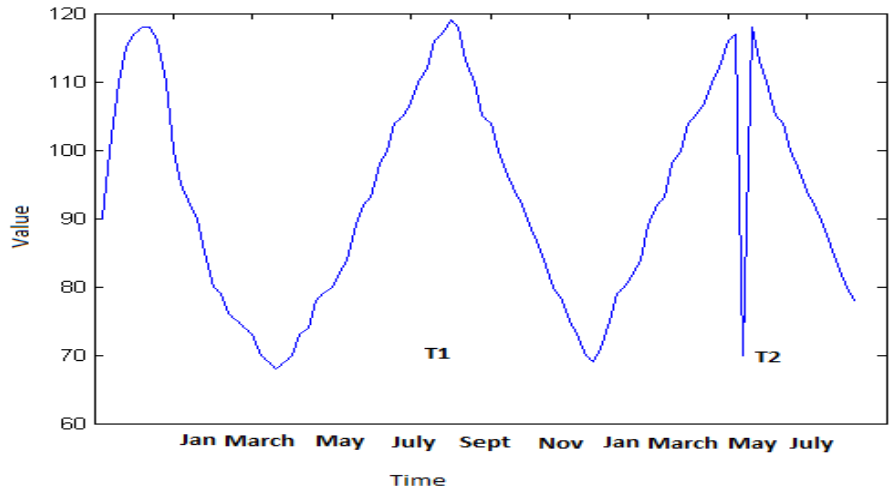

Figure 2.2 One example of contextual anomaly

2.3.3 Output of Anomaly Detection

The output of anomaly detection can be presented in two different manners: *scores* and *labels* (Chandola et al., 2009). The score technique allocates a weight to each data observation, depending on the degree to which it is considered anomalous. Labels are categories assigned to observations, either normal or anomalous. Labels typically take the form of binary classes (anomaly or normal).

2.4 Support Vector Machine

SVM was designed by (Vapnik, 1998) and based on *Structural Risk Minimization(SRM)*, which orders machine learning techniques according to their complexity (Vapnik and Sterin, 1977). SVM converts data to higher-dimensional space via non-linear mapping, in order to compute the optimal separating hyperplane as the proper decision boundary. In short, SVMs can be used for binary or multi-class classification tasks to discover the decision boundary between classes. The SVM algorithm is based on four important concepts: separating hyperplanes, the maximum margin classifier, the soft margin and the kernel function (Noble, 2006).

The separating line is the line that separates the 2D area into two different sub-areas, whereas the separating hyperplane is a plane that separates the 3D space into two. If we can find one separating line or hyperplane between two classes, then we can find many more of them, as shown in figure 2.3. In this figure, the orange and navy points are the examples of two different classes, and the dashed green lines are three possible separating lines that can separate these classes. Among these possible classifiers, there is only one unique *maximum margin classifier*, which has a maximum distance to the observations. Considering the distance of each point to the various separating hyperplanes in the space called a *margin*, the principal aim of this metric is to choose the hyperplane with the maximum margin. Figure 2.4 shows the maximum margin classifier by a solid green line. The points that are pointed out by red circles are SVs, i.e. the points that are located exactly on the margin (Schölkopf and Smola, 2002). More profound explanation of these observations is provided in section 2.5.

In most cases, it is impossible to find a line that can cleanly separate the data into classes, so a *soft margin* is introduced. The soft margin allows some points to violate the selected hyperplane. In fact, the word *soft* is chosen for these margins since they are flexible and allow some training points to be located on the wrong side of the hyperplane or on the margin, as illustrated in figure 2.5. Obviously, it would be unbearable to let plenty of mis-classifications in this method. Therefore, the soft margin uses a parameter called C to control how many

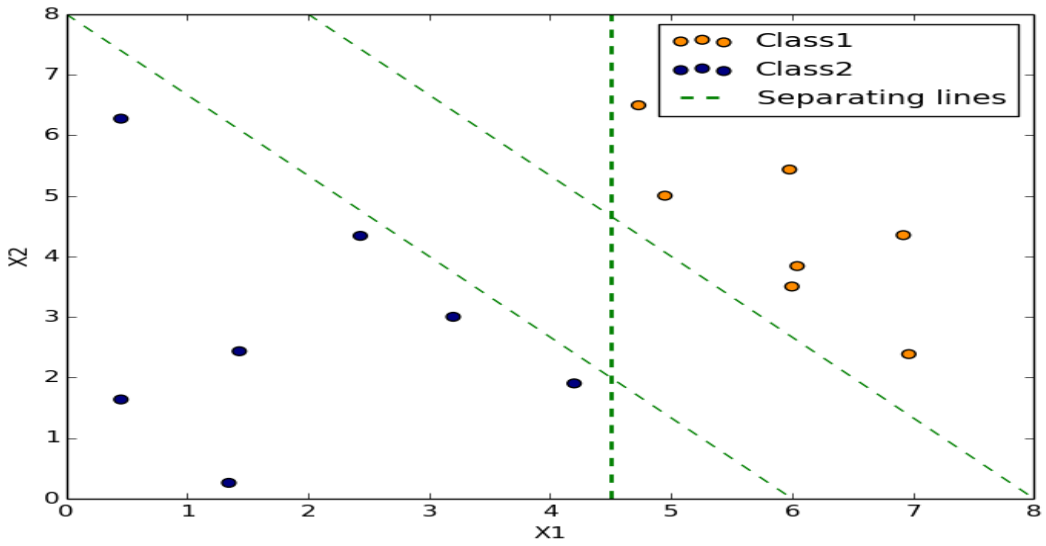

Figure 2.3 2D toy example of binary classification(Possible separating hyperplanes)

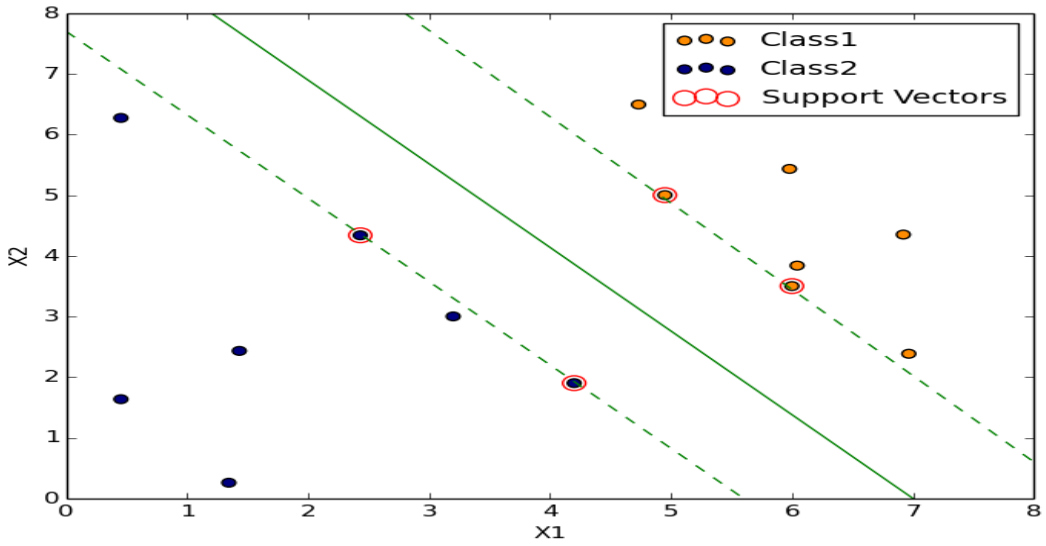

Figure 2.4 2D toy example of binary classification (Maximum Margin classifier)

mis-classifications are allowed and how far from the hyperplane they are permitted to be. C is a variable which depicts the rate of mis-classifications or the percentage of violation, and can be determined through cross-validation. If this tuning parameter has a large value, the

size of the margin is large and vice-versa.

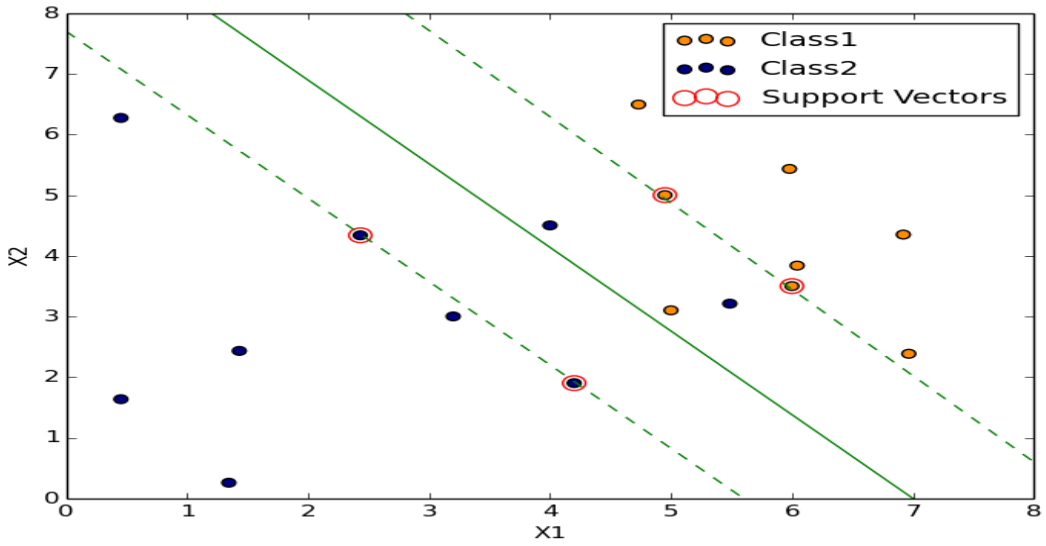

Figure 2.5 2D toy example of binary classification (Soft Margin)

Finding a linear separator for data in low-dimensional space can sometimes be an impossible task. Figure 2.6 shows another toy example of binary classification where there is no way to find a linear classifier for separating two classes. Hence, the projection technique on a higher dimension is employed. In this projection, a hyperplane is found and then projected back in the initial state as shown in figure 2.7. In this figure, it is possible to observe that by adding one dimension to inputs, we can separate two classes by a hyperplane.

Again, observations are presented in vectors, and the main idea behind SVM is to classify non-linear problems using the similarity between vectors. One of the simplest ways to calculate the similarity between two vectors is the *dot product* or *inner product*. Note that in most cases calculating dot product between two vectors in higher dimensions is computationally intensive. For this reason, *kernels* are introduced to be able to calculate the similarity of two vectors without explicitly computing potentially infinite-dimensions. Generally, there are many kernels such as RBF, polynomial, linear and hyperbolic tangent.

2.4.1 Kernels

Kernels are employed to demonstrate the similarities between two observations in higher dimensions (Hastie et al., 2013). Note that any machine learning algorithm that can be

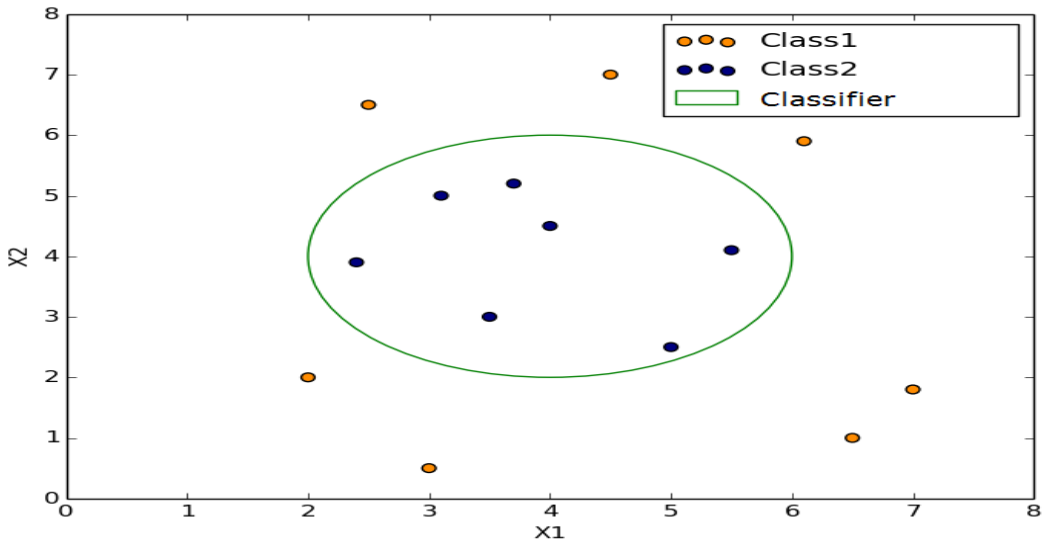

Figure 2.6 2D toy example of binary classification(Non-linear classifier)

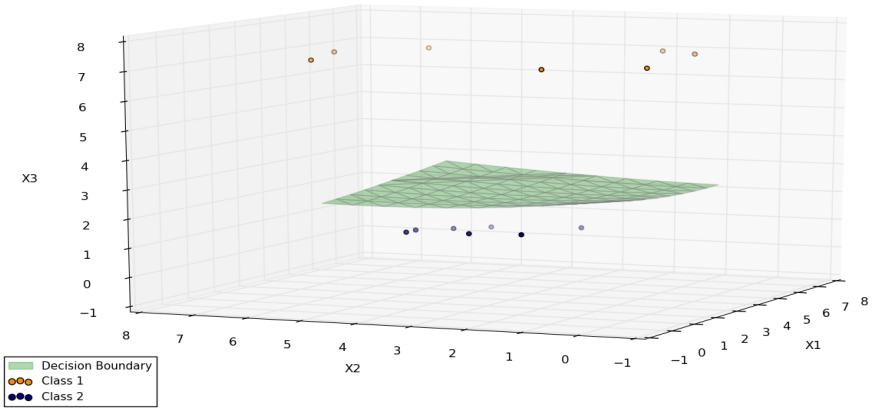

Figure 2.7 3D toy example of binary classification(Non-linear classifier in input space but linear classifier in feature space)

written in dot products can employ kernels. Thus, we can also use kernels in the SVM algorithm. The equation of the kernel function is as follows:

$$K(x_i, x_j) = \Phi(x_i).\Phi(x_j). \tag{2.5}$$

$\Phi : \mathcal{X} \rightarrow \mathcal{H}$ is the mapping function, x_i and x_j are in input space $\mathcal{X}$, and $\Phi(x_i)$ and $\Phi(x_j)$ are the vectors mapped into feature space $\mathcal{H}$ where the dot products are computed, and it can be replaced by the kernel function.

The kernel was proposed by Vapnik (1998), and it is a replacement of the dot product between two vectors. According to equation 2.5, we can compute the dot product of two vectors without directly computing the nonlinear mapping. Specifically, the kernel function can provide us the value of the dot product between two vectors in feature space without exactly calculating this feature space. Note that the dot product between two vectors $\Phi(x_i).\Phi(x_j)$ is a scalar value, and it is always possible to replace the dot product with the kernel; therefore, the output of the kernel is also a scalar value. The idea of replacing $\Phi(x_i).\Phi(x_j)$ by a kernel function is called the *kernel trick* (Tax, 2001). When two classes are not linearly separable, we can take advantage of the kernel by mapping the data to feature space and linearly separating them. In the following sections, we explain the linear kernel, polynomial kernel and RBF kernel. Again, the reason for why we use kernels rather than the regular feature mapping is that they are less computationally intensive.

Linear Kernel

By taking the inner product function 2.5 into account and transforming it into the function 2.6, a linear kernel will be created. This linear function is another form of the support vector classifier.

$$K(\langle x_i, x_k \rangle) = \sum_{j=1}^{p} x_{ij} x_{kj}, \tag{2.6}$$

where p is the number of training examples.

Polynomial Kernel

The polynomial kernel is as follows:

$$K(\langle x_i, x_k \rangle) = (1 + \sum_{j=1}^{p} x_{ij} x_{kj})^d, \tag{2.7}$$

where d is a positive integer value that depicts the degree of the polynomial kernel function.

The polynomial kernel leads to a more flexible decision boundary compared to the linear kernel. If the value of d is equal to one, then this means that we produce the same function of the support vector classifier or linear kernel, whereas if d is greater than one, the data project onto higher dimensions. In the case of applying the polynomial kernel to three dimensions and bringing it back to two dimensions, the hyperplane changes into a circle.

In the case of a polynomial of degree 2 or quadratic kernel, the function 2.7 changes into the following:

$$K(\langle x_i, x_k \rangle) = (1 + \sum_{j=1}^{p} x_{ij} x_{kj})^2 \tag{2.8}$$

Moreover, in the case of the cubic kernel, the function 2.7 changes into the following:

$$K(\langle x_i, x_k \rangle) = (1 + \sum_{j=1}^{p} x_{ij} x_{kj})^3 \tag{2.9}$$

Radial Basis Function Kernel

Radial basis function (RBF) is a symmetric p-dimensional kernel on some special centroids and obeys the following function:

$$K(\langle x_i, x_k \rangle) = exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{kj})^2), \tag{2.10}$$

where γ is the kernel parameter with a positive value. If γ has a large value, then it shows that only observations near each other are considered to be similar. However, if γ has a small value, then the observations located far from each other are considered to be similar. In other words, the γ parameter indicates how far training examples can have an impact on the determination of the test point's label (Hastie et al., 2013).

2.4.2 Kernel example

To better understand the kernel trick, we provide one numerical example. Consider that we have two vectors $s = [2, 4]$ and $u = [3, 6]$ and that we want to project them to six dimensions with the mapping function of $\Phi(s_1, s_2) = (1, s_1, s_2, s_1^2, s_2^2, s_1 s_2 \sqrt{2})$ and calculate the dot product. In this case, these new vectors of $\Phi(s) = [1, 2, 4, 4, 16, 8\sqrt{2}]$ and $\Phi(u) = [1, 3, 6, 9, 36, 18\sqrt{2}]$ are created. Then, the dot product of these two vectors is as follows:

$$\Phi(s).\Phi(u) = 1 + 1 \times 3 + 1 \times 6 + 1 \times 9 + ... + 8\sqrt{2} \times 36 + 8\sqrt{2} \times 18\sqrt{2} = 3025 \qquad (2.11)$$

However, rather than using 2.11, we can use a kernel with the following equation:

$$K(s,u) = (s.u+1)^2 = ([2,4].[3,6]+1)^2 = (2\times 3+2\times 6+4\times 3+4\times 6+1)^2 = 3025 \quad (2.12)$$

In equation 2.11 we have 36 multiplications and 35 summations, whereas in 2.12 there are 5 multiplications and 4 summations. It is clear that equation 2.12 is less computationally intensive compared to equation 2.11. For this reason, we use kernels rather than actually projecting the data to higher dimensions.

The empirical results demonstrate that in most of the training phases, the choice of kernel influences the accuracy and the number of mis-classifications. It is possible to justify its influence through the following example. For instance, to find a proper classifier for some datasets, the polynomial kernel may fail to calculate the proper separating hyperplane, whereas the RBF kernel can find this hyperplane in higher dimensions. Generally, choosing the proper kernel is primarily achieved through trial and error via cross-validation among a set of standard kernel functions.

It should be emphasized that the kernel is the replacement of the mapping function. Precisely, we replace mapping function with the kernel to avoid intensive computation. As such, there is no order in employing mapping function and kernel.

2.5 One-class SVM

Consider that we have some observations with probability distribution P and that we want to estimate a subset S of the input space. This estimation is in a way that the probability of a data point drawn from distribution P not in the subset S is controlled by a parameter called $\nu\epsilon(0,1)$. ν is an important user-defined parameter that shows the upper bound of the fraction of outliers as well as the lower bound of the fraction of SVs. To perform this estimation, we need to find a function f that returns a positive value for S and a negative value for its complement $\overline{S}$. To calculate this function, Schölkopf et al. (2001) proposed an algorithm called one-class SVM. Specifically, they developed an algorithm that can calculate a function f with a value of $+1$ for most of the data points in a "small" area and a value of

-1 for a fraction of data points that are not located in this small area (cf. figure 1.2). This process is possible through two steps: mapping the data to a new feature space with kernels and calculating one optimal hyperplane with the maximum distance from the origin.

Again, in binary classification, if the observations are not linearly separable, then we can take advantage of projecting data to higher dimensions and subsequently separate them by a hyperplane. Note that mapping training examples to higher dimensions is the first step of this algorithm. Thus, we have to map data to higher dimensions. Since mapping data to higher dimensions is computationally intensive, we can use the idea of the kernel as explained in section 2.4. This indicates that employing the kernel is mandatory in the one-class SVM implementation since this is the first step of this algorithm.

One-class SVM is an extension of SVMs to the case of unlabeled data and can resolve outlier detection problems (Schölkopf et al., 2001). In fact, outlier detection is one of the tasks for which this algorithm can be applied. For this purpose, we need to control the fraction of outliers via parameter ν. Note that outlier detection is also known as one-class classification, as mentioned in (Tax, 2001). Recall that in this type of classification problem, only the patterns of one class are well defined. According to Tax's definition, the points in this class are named *target* objects, and any points that are not in this class are considered to be *outliers*.

Schölkopf et al. (2001) designed an algorithm to estimate function f such that its output has positive values for the majority of the data points. In other words, this algorithm seeks a decision boundary with the maximum distance between most of the data points and the origin, as shown in figure 2.8. In this figure, the right plot shows one possible separating line as a one-class classifier before the introduction of kernels, and any point that falls beyond this line has an output of -1. The left plot shows one possible separating hyperplane after the introduction of kernels, leading to a small area with an output of 1 for target observations and an output of -1 for non-target or outlier observations. Note that in the binary classification setting, we refer to the points either on the wrong side of the hyperplane or on the margin as *margin errors* (note that the good side is the side that the observation of one specific class should be located in its true side). However, in one-class classification, these margin errors are known as outliers (Schölkopf et al., 2001).

This algorithm detects non-target observations based on this one class, and any observation that deviates from the target observations is considered to be an outlier. The algorithm chooses a hyperplane as a decision boundary that distinguishes a small section of the data points from a large section of data points. This small part of the data is marked as outliers, and the large part is considered to be the safe area. In right plot of figure 2.8, the majority

of the training observations are placed inside the calculated decision boundary, and there are some outliers between the decision boundary and the origin.

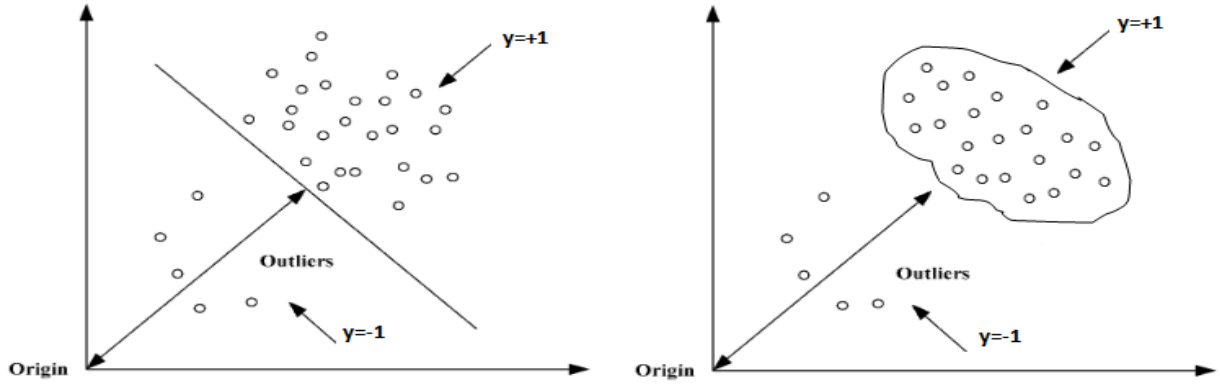

Figure 2.8 One-class SVM hyperplane without the introduction of kernels (left), and one-class SVM hyperplane with the introduction of kernels (right)

In the following section we present one-class SVM algorithm. Moreover, there is one numerical example in appendix A that we explain how the outliers are detected.

2.5.1 Algorithm

Consider $\{x_1, ..., x_l\} \epsilon \mathcal{X}$ as the training data, where $l \epsilon N$ is the number of observations and $\mathcal{X} \epsilon \mathbf{R}^N$ shows some nonempty set.

Recall that we want to find a function that takes a value of +1 for a small area that has the majority of the training examples and a value of -1 for any point not in this small area. For this purpose, we need to map the training points to higher dimensions via a kernel and calculate the hyperplane that separates them from the origin with the maximum distance. The hyperplane can be written as follows:

$$\{x \epsilon \mathcal{X} | (\mathbf{w}, x) - \rho = 0\}, \quad \mathbf{w} \epsilon \mathcal{X}, \rho \epsilon \mathbf{R}, \tag{2.13}$$

where $\mathbf{w}$ is a vector orthogonal to the hyperplane and ρ is the offset.

As mentioned in section 2.4.1, $\Phi : \mathcal{X} \rightarrow \mathcal{H}$ is a feature map from the input to feature space, and the kernel function is as follows:

$$k(x_i, y_j) = (\Phi(x_i).\Phi(y_j)). \tag{2.14}$$

The distance between the origin and the hyperplane is shown by $\frac{\rho}{\|\mathbf{w}\|}$ ($\| \mathbf{w} \|$ is the magnitude or norm 2 of vector w), and maximizing this distance is equal to minimizing $\frac{\|\mathbf{w}\|^2}{2}$. Note that small values of $\| \mathbf{w} \|$ indicate a large separation distance from the origin. Moreover, we need to allow some observations to fall beyond the hyperplane, leading to a smooth decision function. Therefore, to find a decision boundary that maximizes the distance between the origin and the majority of the data points, we need to solve the following primal objective function:

$$min_{w,\xi,\rho} \frac{\| \mathbf{w} \|^2}{2} - \rho + \frac{1}{\nu l} \sum_{i=1}^{l} \xi_i \tag{2.15}$$

$$\text{Subject to:} \quad \mathbf{w}\Phi(x_i) \geq \rho - \xi_i, \quad \xi_i \geq 0. \tag{2.16}$$

Here, $\xi = \{\xi_1, ..., \xi_n\}$ is a vector of *slack variables* that allow for mis-classification. The value of ξ_i illustrates the location of each ith observation with respect to the hyperplane. The closer its value is to zero, the better is the classification result. If $\xi_i = 0$, then the ith observation is on the right side of the decision boundary, and if $\xi_i > 0$, then the ith observation is on the wrong side of the hyperplane. $\nu \epsilon (0, 1]$ is the outlier rate or the regularization parameter that is an upper bound on the fraction of outliers and also a lower bound on the fraction of SVs. Nonzero ξ_i are penalized via equation 2.15, and we can control them by the value of parameter ν. Large values of ν lead to a higher upper bound for the fraction of outliers and to a greater possibility of having mis-classifications. Conversely, small values of ν lead to a smaller upper bound and to a lower possibility of having mis-classifications. Hence, the first part of this primal objective function is to minimize ε_I error, the second part is to minimize ε_{II} error, and the fraction of $\frac{1}{\nu l}$ acts as a regularization parameter.

To better understand the aim of this algorithm, consider the 2D toy example in figure 2.9, where the orange points are the target objects and the navy point is the only outlier. Again, we call it an outlier or non-target pattern since it is on the wrong side of the decision function. Here, the hyperplane is shown by a green line that separates all the target points from the origin except the outlier. The slack variable for this point is larger than zero ($\xi_i > 0$). The distance between the outlier and the frontier is equal to $\frac{\xi}{\|\mathbf{w}\|}$.

Recall that the observations that impact the computed classifier are those that lie exactly on the margin, i.e., SVs. Thus, the training examples on the correct side of the one-class

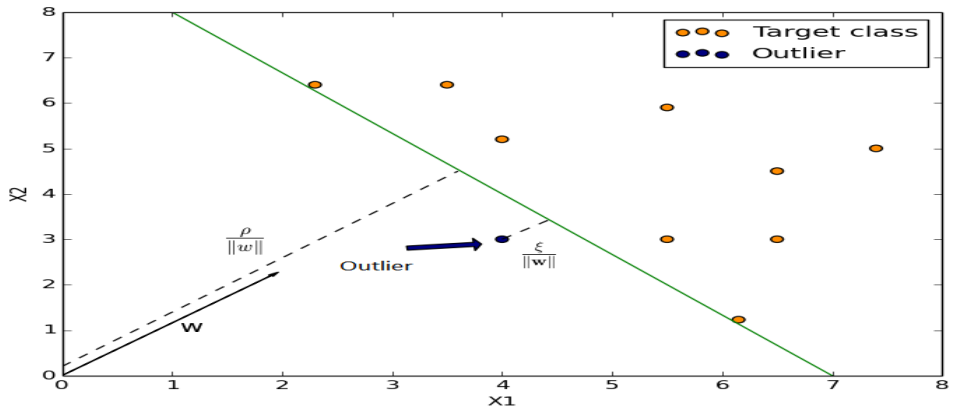

Figure 2.9 A 2D toy example showing one-class SVM hyperplane

classifier do not affect the calculated separating hyperplane (the observations with $\xi_i = 0$).

If $\mathbf{w}$ and ρ are able to solve the quadratic problem, then the following decision function is obtained:

$$f(\mathbf{x}) = sgn(\mathbf{w}.\Phi(x) - \rho). \tag{2.17}$$

.

The function 2.17 acts as a decision boundary, is a sign function that returns a positive value for majority of the data points, and it can be controlled by parameter ν. In fact, a test example x^* is assigned a class based on the output of $f(x^*)$.

The primal objective function formulated in 2.15 can be changed to dual form to have simpler constraints and also to have the dot product form. In fact, in learning with kernels, we need to have dot products in our optimization formula; therefore, we transform the function from primal to dual (Schölkopf and Smola, 2002). This transformation from primal to dual is possible through Lagrange multipliers $\alpha, \beta \geq 0$ and a Lagrangian, as follows:

$$\begin{aligned} L(\mathbf{w}, \xi, \rho, \alpha, \beta) = \frac{\| \mathbf{w} \|^2}{2} + \frac{1}{\nu l} \sum_{i=1}^{l} \xi_i - \rho \\ - \sum_{i=1}^{l} \alpha_i (\mathbf{w}.\Phi(x_i) - \rho + \xi_i) - \sum_{i=1}^{l} \beta_i \xi_i. \end{aligned} \tag{2.18}$$

.

Equation 2.18 is a quadratic programming problem. Note that the Lagrange multipliers are

the coefficients of the kernel expansion in this algorithm. To compute the minimum of the Lagrangian L, it is necessary to take the derivative with respect to $\mathbf{w}$ and ρ and then set it to zero, yielding the following:

$$\mathbf{w} = \sum_i \alpha_i \Phi(x_i) \tag{2.19}$$

$$\alpha_i = \frac{1}{\nu l} - \beta_i \leq \frac{1}{\nu l}, \sum_i \alpha_i = 1. \tag{2.20}$$

In equation 2.19, all data points with $\alpha_i > 0$ are SVs. Again, SVs are the points that lie either on the margin or on the wrong side of the hyperplane, and they are the points that support the classifier. Any other points are considered as irrelevant. Using equation 2.19, the decision function 2.17 changes to the following kernel expansion:

$$f(\mathbf{x}) = sgn\left(\sum_i \alpha_i k(\mathbf{x}_i, \mathbf{x}) - \rho \right). \tag{2.21}$$

.

By substituting 2.19 and 2.20 and using kernel equation 2.14 in Lagrangian L, the dual problem is obtained as in equation 2.22.

$$\begin{aligned} min_\alpha \frac{1}{2} \sum_{ij} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) \\ \text{subject to: } 0 \leq \alpha_i \leq \frac{1}{\nu l}, \quad \sum_{i=1}^{n} \alpha_i = 1. \end{aligned} \tag{2.22}$$

Thus, the value of parameter ρ can be calculated as follows:

$$\rho = \sum_j \alpha_i k(\mathbf{x}_j, \mathbf{x}_i). \tag{2.23}$$

To conclude this section, note that this optimization problem is a convex optimization, and it yields an optimal solution.

2.5.2 Optimization

To compute dual coefficients, we can take advantage of quadratic programming. Schölkopf implemented their algorithm via *sequential minimal optimization (SMO)*, which divides the

optimization of equation 2.22 into the smallest optimization tasks. It is possible to solve these optimization tasks analytically, meaning that there is no need to perform quadratic optimization tasks in the inner loop of this algorithm. It is not possible to perform optimization on individual variables without violating the constraints of this optimization function. Hence, the optimization is performed on pairs of variables rather than on one variable.

Elementary optimization step. Imagine that we want to optimize over α_1 and α_2 without any changes in the other variables. In this case, equation 2.22 changes to the following ($K_{i,j}$ is shorthand for $K(\mathbf{x}_i, \mathbf{x}_j)$):

$$min_{\alpha_1,\alpha_2} \frac{1}{2} \sum_{i,j=1}^{2} \alpha_i \alpha_j K_{i,j} + \sum_{i=1}^{2} \alpha_i C_i + C, \tag{2.24}$$

where $C = \sum_{i,j=3}^{l} \alpha_i \alpha_j K_{i,j}$ and $C_i = \sum_{j=3}^{l} \alpha_j K_{i,j}$, with the following constraints:

$$0 \leq \alpha_1, \alpha_2 \leq \frac{1}{\nu l}, \sum_{i=2}^{2} \alpha_i = \Delta, \tag{2.25}$$

with $\Delta = 1 - \sum_{i=1}^{l} \alpha_i$. After substituting and taking the derivative with respect to α_2 and setting it to zero, we obtain the following function for calculating the α_2 parameter:

$$\alpha_2 = \frac{\Delta(K_{11} - K_{12}) + C_1 - C_2}{K_{11} + K_{22} - 2K_{12}}. \tag{2.26}$$

Through calculating α_2, the value for α_1 can be computed using the following equation:

$$\alpha_1 = \Delta - \alpha_2. \tag{2.27}$$

After each step for calculating α_2 (equation 2.26) and α_1 (equation 2.27), we update variable ρ.

If we rewrite equation 2.26 in terms of the outputs of the kernel extension, then we can better understand how the parameters are updated. Considering α_1^* and α_2^* as the previous values for the Lagrange parameters, we obtain the output of the kernel expansion as follows:

$$O_i = K_{1i}\alpha_1^* + K_{2i}\alpha_2^* + C_i, \tag{2.28}$$

as the corresponding outputs for each training example. By eliminating C_i in function 2.28, we obtain the following equation for calculating α_2:

$$\alpha_2 = \alpha_2^* + \frac{O_1 - O_2}{K_{11} + K_{22} - 2K_{12}}. \tag{2.29}$$

Note that this elementary optimization step can be performed on every pair of these Lagrangian parameters.

Initialization of the algorithm. We initialize the value of α_i to $\frac{1}{\nu l}$ for a random v fraction of training points. Additionally, we initialize the parameter ρ to $max\{O_i : i\epsilon[l], \alpha_i > 0\}$.

Optimization algorithm. For the optimization, we select a variable in two possible ways, as follows. In this optimization algorithm, SV_{nb} stands for the indices of variables with $0 < \alpha_i < \frac{1}{\nu l}$. When the optimization terminates, this set shows the indices of SVs.

1. All the training points are checked until we find one observation that violates the *Karush-Kuhn-Tucker (KKT)* conditions (2.31). Once we choose α_i, we are able to select α_j according to the following:

$$j = argmax_{n\epsilon SV_{nb}} \|O_i - O_j\|. \tag{2.30}$$

2. This is the same as 1; however, scanning is only performed on components of the SV_{nb} set.

In the implementation of this optimization algorithm, checking 1 is followed by checking 2 until we find no KKT violators in the SV_{nb} set. After this process, the control returns to checking 1 until no KKT violators are found. At this state, we can terminate the optimization. Any indices in the SV_{nb} set show the index of a training point that has a strong influence on the decision boundary.

It is necessary to consider KKT conditions (Bertsekas, 1999) when solving quadratic programming problems. These conditions are constraints on Lagrange multipliers, as shown in the following equations:

$$\begin{aligned} (O_i - \rho).\alpha_i > 0 \\ (\rho - O_i).(\frac{1}{\nu l} - \alpha_i) > 0. \end{aligned} \tag{2.31}$$

After calculating the optimal values for parameters α_i, we can update the value of ρ (equation 2.23); finally, we are able to compute the decision function for all the test examples (equation 2.21). It should be emphasized that some of the presented equations are used for outlier

computation as explained step by step in appendix A.

2.5.3 Parameters

In general, one-class classification methods have two types of parameters, called *free* and *magic* parameters. A free parameter is a parameter that is optimized automatically in the algorithm, whereas a magic or hyper-parameter is a user-defined parameter. In the one-class SVM algorithm, we have l number of free parameters called α_i (dual coefficient), and they are optimized through the optimization algorithm explained in section 2.5.2. Recall that l is the number of training observations; therefore, the total number of free parameters for this algorithm is l. Moreover, we have one magic parameter called ν that specifies the fraction of rejected observations by the decision boundary.

If we choose the RBF kernel as the similarity function between observations, we need to adjust the γ parameter. We assume that both ν and the γ parameter have an impact on the calculated decision boundary (Schölkopf et al., 2001). We investigated the influences of ν and γ on the 2D spherical-shaped toy example. The spherical toy example has 200 observations, in which ten observations are non-target and the remaining observations are target observations. The scatter plot of this toy example is plotted in 2.10. We can observe that the non-target observations (red points) are distributed around the target observations (green points).

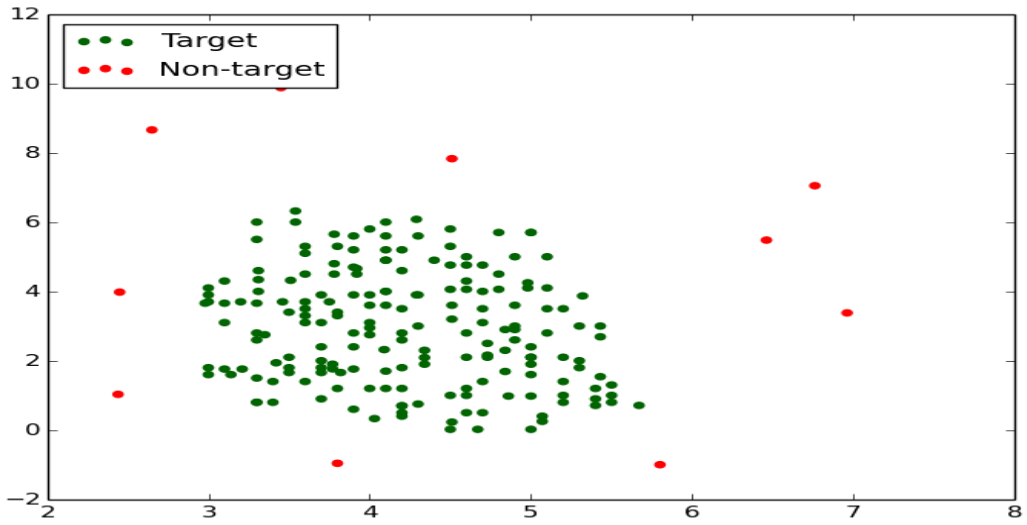

Figure 2.10 Scatter plot of spherical 2D toy example

Figure 2.11 presents four different plots with four different γ: $\gamma = [0.1]$, $\gamma = [0.5]$, $\gamma = [1]$ and $\gamma = [5]$. Note that in all of these plots, the value of ν is equal to 0.05. Here, the red line is the calculated frontier, and the algorithm considers five percent of the training observations to fall outside of the decision boundary. Larger γ values lead to a tighter classifier, as shown in plots of $\gamma = 5$ since only the observations that are close to each other are considered similar. Therefore, with increasing γ, the number of SVs increases. Moreover, having large γ values may lead to a higher FP rate (Schölkopf et al., 2001).

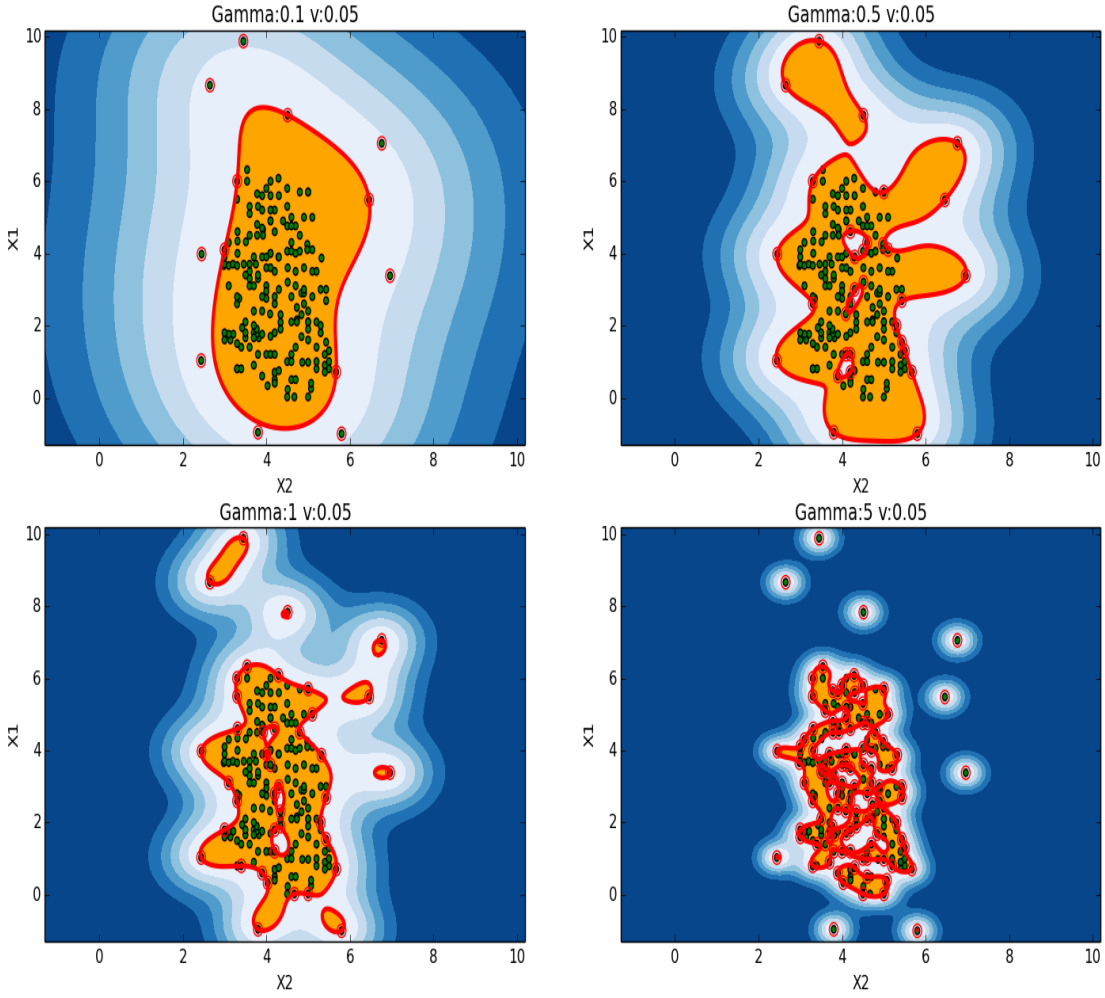

Figure 2.11 Spherical 2D toy example with $\nu = 0.05$ and four different γ values

Figure 2.12 shows four different plots with a fixed value of $\nu = 0.20$ and the same γ values as in figure 2.11, allowing the one-class SVM algorithm to consider some observations as non-target examples with an upper bound of 20 percent.

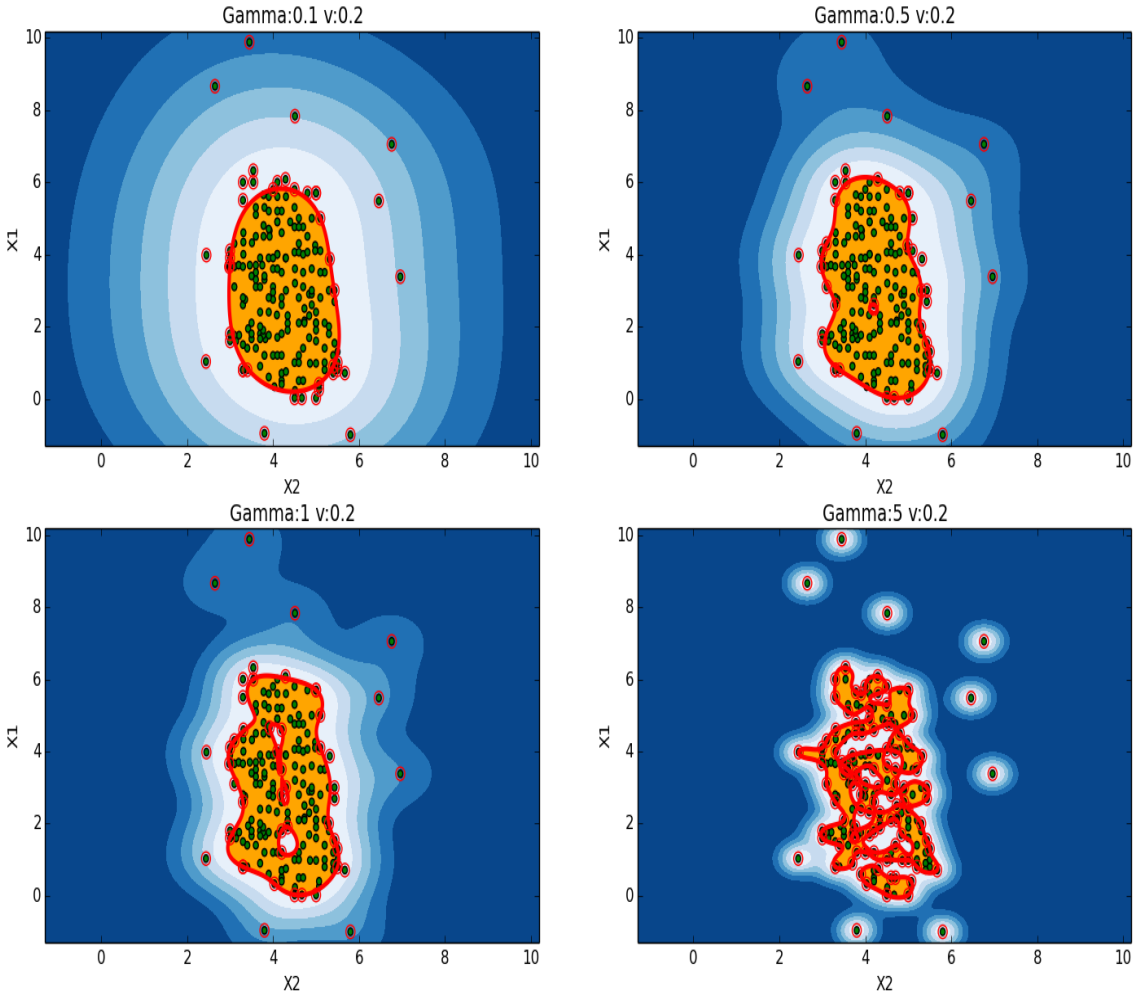

Figure 2.12 Spherical 2D toy example with $\nu = 0.2$ and four different γ values

Figure 2.13 presents four different plots with a fixed value of $\nu = 0.5$ and the same γ values as in figure 2.11, allowing the one-class SVM algorithm to consider some observations as non-target examples with an upper bound of 50 percent.

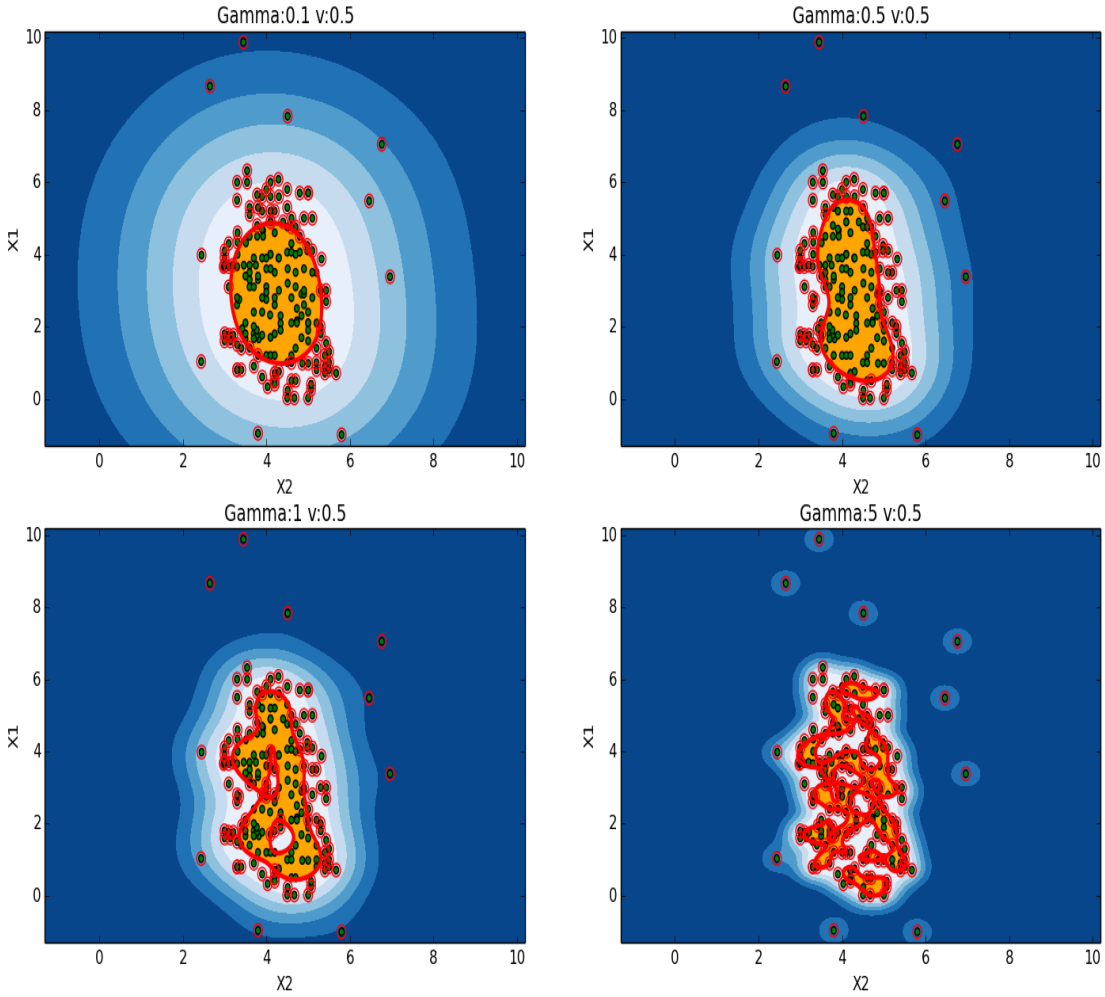

Figure 2.13 Spherical 2D toy example with $\nu = 0.5$ and four different γ values

After closely examining the 12 different plots presented in figures 2.11, 2.12 and 2.13, we can conclude that small values of γ are better classifiers since they are not overfitted. Moreover, large values of ν may lead to an incorrect one-class classifier. This result is because parameter ν shows a lower bound of the fraction of SVs, and by increasing this value, a larger fraction of training patterns should be considered as SVs. Meanwhile, for the task of outlier detection, the value of parameter ν should be considered small. Thus, we tend to choose small values for parameter ν (Schölkopf et al., 2001). However, the best possible values for these two parameters can be calculated through K-fold cross-validation. This technique is a model selection method, and it divides the data into K segments, each time using K-1 sections to train and one section for validation. This process of training and validating should be performed K times for each model, leading to K average values. After calculating these average values, we choose the lowest value as the best model.

Table 2.1 summarizes the fraction of SVs, fraction of outliers and training time for different possible values of parameter ν ($\nu \epsilon \{0.05, 0.1, 0.2, 0.5\}$). Accordingly, table 2.2 summarizes the fraction of SVs, fraction of outliers and training time for different possible values of parameter γ ($\gamma \epsilon \{0.1, 0.5, 1, 5\}$).

Table 2.1 Experimental results for different values of parameter ν

ν	Training time (s)	fraction of SVs	fraction of Outliers
0.05	0.0011	0.16	0.09
0.1	0.0027	0.16	0.11
0.2	0.0028	0.22	0.18
0.5	0.0044	0.52	0.5

Table 2.2 Experimental results for different values of parameter γ

γ	Training time (s)	fraction of SVs	fraction of Outliers
0.1	0.0012	0.22	0.2
0.5	0.0029	0.22	0.18
1	0.0037	0.25	0.18
5	0.0085	0.53	0.31

According to the experimental results shown in tables 2.1 and 2.2, we can conclude that by increasing the values of parameters γ and ν, the computation time also increases. This increase in computation time is due to the increase in the number of SVs. Since parameter ν is the lower bound of the fraction of SVs, the number of SVs increases to satisfy this constraint.

In his paper, Schölkopf (Schölkopf et al., 2001) only used the RBF kernel without mentioning the polynomial kernel. However, we investigate the result of the polynomial kernel on this toy example. If we choose the polynomial kernel, we need to evaluate different values for the degree of the kernel ($d = 2, 5, 10, 20$).

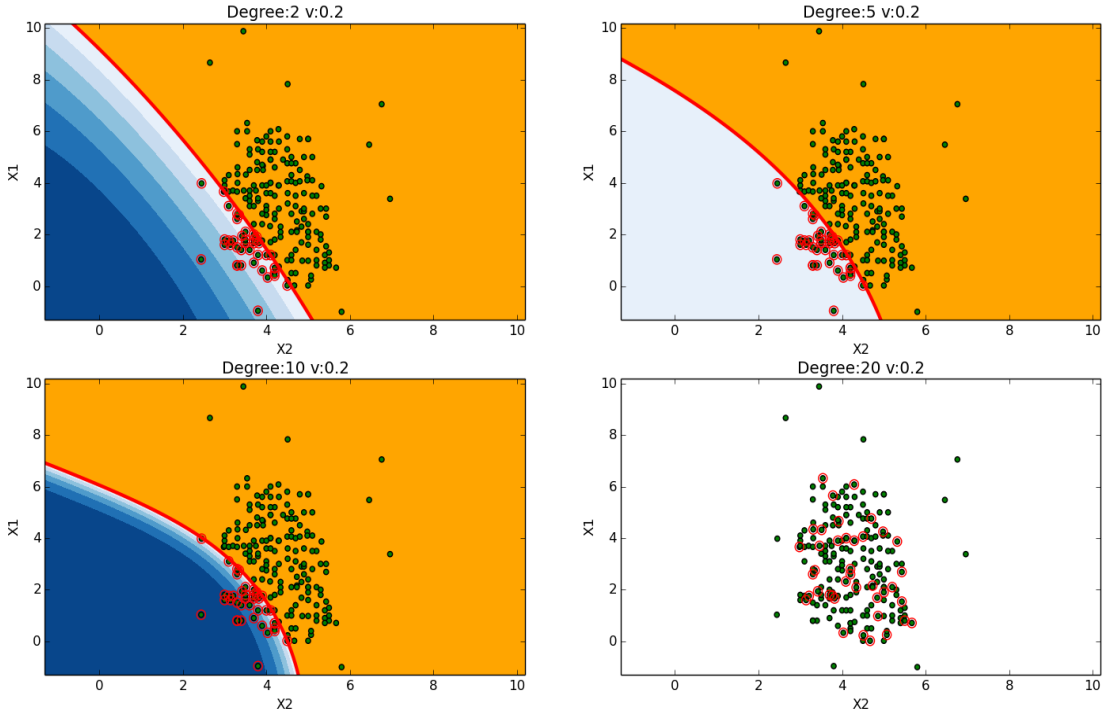

Figure 2.14 Scatter plot of banana-shaped 2D toy example with polynomial kernel

Figure 2.14 suggests that the polynomial kernel fails to detect a proper one-class classifier. As explained in Tax (2001), when the data are not centered around the origin, the values of the vector's components become large when applying polynomial kernels, leading to very small angles between vectors. Consequently, the patterns with large norm values suppress the remainder of the patterns. Meanwhile, Tax suggested that even if we transform the data to zero mean and unit variance, we will end up again having a frontier influenced by the large norms. Hence, we are not able to use this kernel for one-class SVM because it fails to detect the proper decision function. Thus, in the remainder of this thesis, we only employ the RBF kernel for this algorithm.

In this chapter, we presented the state of the art in one-class classification algorithms and in false alarm reduction methods. Then, we explained intrusion detection systems. Afterwards, we discussed SVM and one-class SVM algorithm.

CHAPTER 3 FALSE ALARM REDUCTION METHOD

This chapter discusses the proposed false alarm reduction method.

3.1 Proposed false alarm reduction method

Our problem is to decrease the false alarm rate in the anomaly-based intrusion detection system. We propose using one-class SVM to detect outliers and SVs, and we use these two sets of observations to decrease the false alarm rate with the same FN rate. Recall that outliers are the observations that fall outside the decision boundary, and SVs are the observations that support the decision boundary (Schölkopf and Smola, 2002). Note that these two sets of observations are not the same, and there are some observations that belong to both sets. In fact, we use the one-class SVM algorithm to solve a one-class classification problem, and we suggest using the output of this algorithm to decrease the false alarm rate. As mentioned in section 2.5, this algorithm wants to estimate a function f that is positive on S and negative on the complement $\overline{S}$. Recall that S is a subset of training observations such that the probability that a test observation drawn from P (the underlying distribution of data is P) not in the subset S is bounded by regularization parameter ν.

First, we need to calculate a binary function f (decision function) based on the one-class SVM algorithm with a proper value for parameter ν and a proper kernel. We can use K-fold cross-validation to tune these two hyper-parameters. Recall that parameter ν is the upper bound for the fraction of outliers, and its value should be greater than zero. For example, if we consider $\nu = 0.1$, then up to 10 percent of the training points can be assumed to be outliers.

We define the set of SVs as $S_{SupportVectors}$, and we also define the training examples with label -1 (not in the subset S) as outlier observations, which we call $\overline{S}_{Outliers}$. We suggest removing the outliers from the set of SVs, leading to a new set called $S_{SafeSVs}$. This set is the SVs that are not located outside the decision boundary, i.e., the observations that are not rejected by the classifier. Furthermore, if we use the calculated decision function f on the unseen observations (test examples), we ultimately have some observations with a value of -1 and some with a value of 1. We define the test examples with a label class of -1 as $\overline{S}_{ProbableAttack}$, and we call them *probable attacks*. In other words, probable attacks are the unseen observations that fall outside the decision boundary.

We suggest comparing the $S_{SafeSVs}$ set with the $\overline{S}_{ProbableAttack}$ set based on their similarity.

This comparison is based on the Euclidean distance between two vectors, and the Euclidean distance (cf. equation 3.1) is used for evaluating the similarity between two vectors. However, the similarity between two vectors should be determined through the proper threshold, and we can use a K-fold cross-validation on the training examples for this purpose. We denote this threshold as T. An observation in the $\overline{S}_{ProbableAttack}$ set will be considered similar with another observation in the $S_{SafeSVs}$ set if it has a Euclidean distance that is less than this calculated threshold. Hence, if an observation has a value that is less than this threshold, then we remove this observation from the $\overline{S}_{ProbableAttack}$ set.

$$d = |x - y| = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2} \tag{3.1}$$

Precisely, after having determined three sets of $S_{SupportVectors}$, $\overline{S}_{Outliers}$ and $\overline{S}_{ProbableAttack}$ and having tuned the threshold parameter, the false alarm reduction method is as follows:

- We subtract the set of $\overline{S}_{Outlier}$ from $S_{SupportVectors}$, leading to a subset of SVs with label 1, and we denote this new set as $S_{SafeSVs}$. In fact, with this subtraction of sets, we have only the positive SVs that have an impact on the calculated decision function.
- We compare all observations in $\overline{S}_{ProbableAttack}$ using the set of $S_{SafeSVs}$ with the calculated value of T. If an observation in the $\overline{S}_{ProbableAttack}$ set has a Euclidean distance value that is smaller than the threshold, then we remove that observation from this set.
- Any observation that remains in the $\overline{S}_{ProbableAttack}$ set is copied in the $\overline{S}_{Attack}$ set, leading to a set of real attacks.

Hence, $\overline{S}_{Attack}$ contains observations with the attack label. Specifically, if we remove similar observations of safe SVs from probable attacks, then the remaining points are considered to be real attacks.

To conclude, the trick that we have performed is that we used SVs with a positive output to decrease the false alarm rate. In this case, any unseen observations that are near these SVs and placed outside the decision boundary are assumed to be non-attack observations.

3.2 Adjusting Parameter T

As explained in the previous section, we have to choose a proper value for parameter T since it is the threshold of how similar two vectors are considered to be. Having large values

for this parameter leads to changing the result, and it may violate the TP rate since even non-similar vectors will be considered the same. Consequently, we tend to choose smaller values for this parameter. However, if we choose very small values showing that two vectors have exactly the same component values, then we may ultimately have no modification in the false alarm rate. Hence, the choice of the value for this parameter can be addressed similarly to a hyper-parameter. For this purpose, we use a K-fold cross-validation technique to adjust parameter T, and the choice of the value for this parameter is based on the value of the ROC score. We employ the K-fold cross-validation (explained in section 4.3) method to ensure that the value that we choose for this parameter can generalize perfectly to unseen observations. As thoroughly explained in section 1.2, the AUC value is used for comparing classifier outputs. Thus, the best model is the one with the highest average AUC value. Note that we may have multiple possible values for parameter T with the highest average AUC. In these situations, for simplicity, we assume that the best choice is the lowest value of these possible choices.

Thus, the following hypotheses should be verified:

- H0a: Large values of parameter T violate the TP rate
- H0b: The transformation methods (cf. explained in section 4.1) of training patterns have an impact on the possible values for parameter T.
- H0c: Among the possible values for parameter T, the one with the highest AUC value is the best choice.

Assumption H0a suggests that we should not use large values for parameter T since we ultimately decrease the rate of detecting real attacks (TP rate). Moreover, H0a is rejected if the data is not scaled. However, since we select the best value for this parameter based on the average ROC score, the selected value will not be large, leading to violating the TP rate as stated in H0a.

Assumption H0b has an influence on the choice of the upper bound value for parameter T. If we use feature scaling to $[0, 1]$, then the calculated Euclidean distance values will be smaller compared to feature scaling to $[0, 10]$ since the norm values of the vector are smaller. Thus, it is suggested that we use smaller values as an upper bound for this parameter. However, if we use the robust scalar method for scaling, then the possible Euclidean distance values become larger values. In this case, it is suggested to use a larger value as the upper bound for this parameter. Furthermore, hypothesis H0c is always true since AUC shows the performance

of the classifier, and we want to choose the best value for parameter T such that it does not impact the final result.

Note that the proper value for this parameter is highly related to the preprocessing method.

3.3 Toy Example

To better describe the proposed method, consider one artificial data set with 50 training examples, in which 20 percent of them are attack (non-target) observations and the remainder are non-attack (target) observations. In this example, each observation is shown by a vector of two components. Furthermore, consider $y\epsilon\{-1, 1\}$ showing the possible class labels of these observations. Note that in one-class SVM, we only use the inputs of observations, and the outputs are used in model selection, i.e., adjusting the values of hyper-parameters. Additionally, imagine that we have 10 test examples to evaluate the classifier using the proposed false alarm reduction method.

For convenience, suppose that we calculate the optimal value of hyper-parameter $\nu = 0.2$, and also consider that we employ the RBF kernel with $\gamma = 0.01$. Furthermore, assume that we use no scaling method. Having determined the value of two hyper-parameters, we can determine the value of parameter T using the 5-fold cross-validation technique. For this purpose, we test different values for parameter T, starting from 0.1 to 3 with a step value of 0.1. Figure 3.1 shows the calculated ROC score values versus the possible values of parameter T. We select the best value for parameter T based on the highest ROC score. Consequently, we choose the value 0.1 as the best value for parameter T. Note that this choice is based on the average ROC score, and we have the highest ROC score with a value of 0.974 for the selected threshold. As shown in the figure, the calculated ROC score becomes smaller for larger values of parameter T. This result suggests that assumption H0a is valid. Moreover, since we use no scaling on the training examples, we use an upper bound with a value of 3 based on assumption H0b.

Figure 3.2 shows the scatterplot of the training examples, where the green points are the target patterns and the red points are non-target instances. Our objective is to find a classifier that places the positive examples in an enclosed area with the maximum distance to the origin and places the negative examples outside this enclosed area.

Suppose that we train the model with proper ν and γ values. The computed value of the intercept is -4.18, and the coefficients of the decision boundary are as follows:

$[1, 1, 0.656, 1, 1, 1, 0.624, 1, 1, 1, 0.72]$

Note that the number of SVs is equal to the number of coefficients. In fact, among these 50

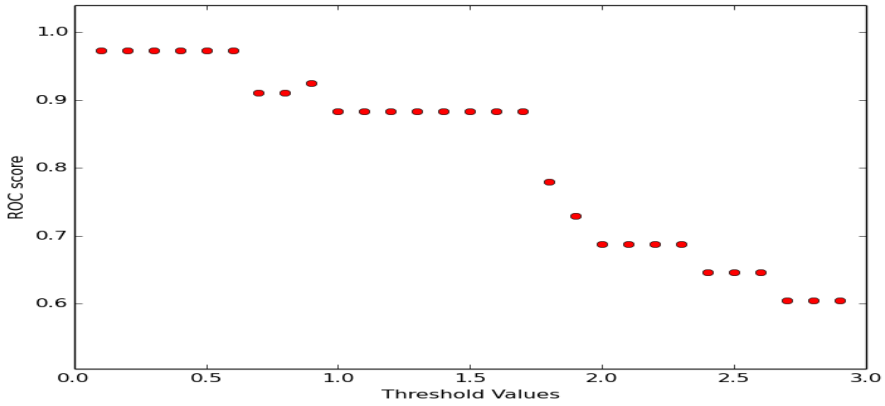

Figure 3.1 ROC score versus parameter T

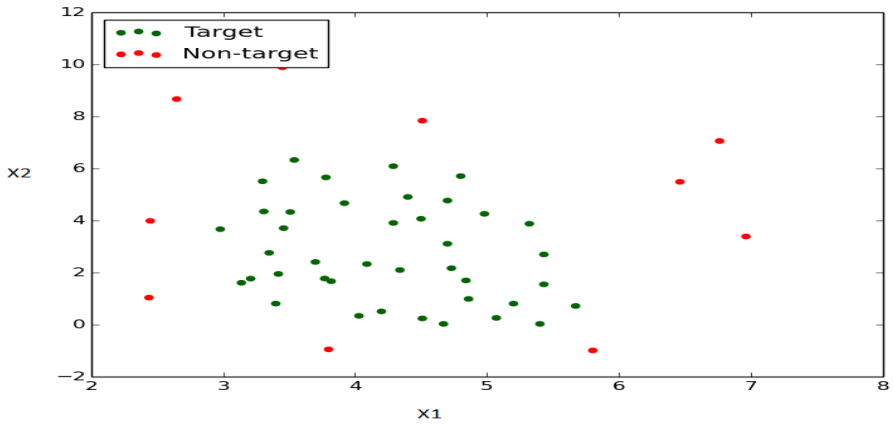

Figure 3.2 Scatterplot

observations that we use for training, only 11 of them are used for calculating the one-class classifier (cf. equation 2.21), and the remaining observations are irrelevant.

Once the model is trained and the set of SVs is calculated, we can calculate the set of outliers. Any training observation that has a negative output based on equation 2.21 is considered to be an outlier. Based on this toy example, the one-class SVM algorithm produces a set of SVs that support the decision function. Among these SVs, nine of them are outliers since they fall outside the computed classifier. In other words, these nine observations are rejected by the decision function.

Figure 3.3 shows the decision boundary on this toy example. The red line is the decision boundary, the green points are positive patterns, the SVs are training patterns with red circles around them, and the white points are the unseen observations that are classified as

attacks, i.e., non-target examples.

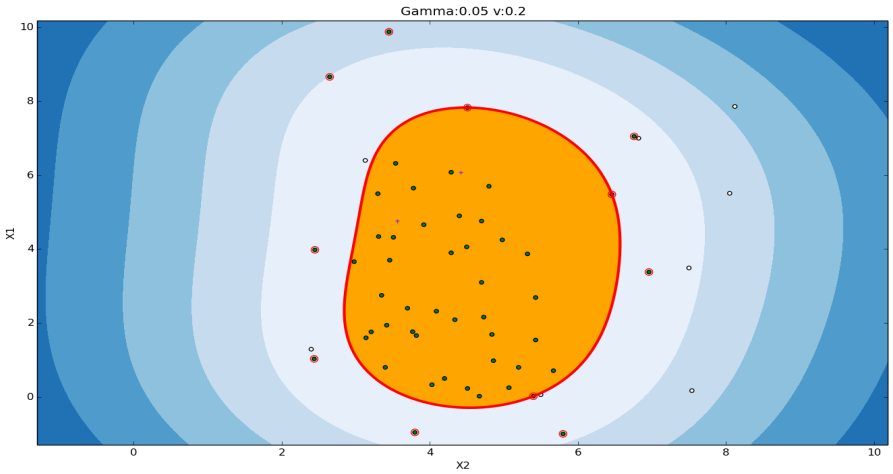

Figure 3.3 Scatterplot with decision boundary with one-class SVM

If we compute the outputs based on this computed classifier, we obtain a set of observations that are classified as non-target examples ($\overline{S}_{Outlier}$). Moreover, if we classify all the testing examples using this classifier, then there will be some observations classified as attacks. We assign them to the set denoted as $\overline{S}_{ProbableAttack}$. After tuning the proper value for parameter T to the value of 0.1, the proposed false alarm reduction method is as follows:

1. In this step, we need to subtract the $\overline{S}_{Outlier} = \{4, 7, 12, 15, 24, 25, 32, 37, 42\}$ set from the $S_{SupportVectors} = \{7, 11, 12, 15, 24, 25, 31, 32, 37, 39, 42, 49\}$ set, yielding the $S_{SafeSVs} = \{31, 49\}$ set. Note that these numbers are the indices of the vectors.

2. We test the trained model on the unseen observations to calculate the $\overline{S}_{ProbableAttack}$ set.

3. We compute the Euclidean distance between each of these patterns in the $\overline{S}_{ProbableAttack}$ set and the patterns in the $S_{SafeSVs}$ set based on equation 3.1. If an observation in the $\overline{S}_{ProbableAttack}$ set is near an observation in the $S_{SafeSVs}$ set, then we can change its label from attack to non-attack. In other words, if the Euclidean distance between two observations is less than the calculated threshold, then we consider it to be a non-attack, leading to reducing the false alarm rate. Table 3.1 shows the calculated Euclidean distance values. Since there are seven probable attack patterns and two safe SV observations, 14 different Euclidean distance values are computed. Only the observation with a Euclidean distance value of 0.099 is lower than 0.1, and we can

change its class label from attack to non-attack. In figure 3.4, we can observe that two observations are members of the $S_{SafeSVs}$ set and that there is one observation close to one of these points, marked by a black diamond point. This unseen observation is outside the enclosed decision boundary; however, it is close (based on Euclidean distance) to one of the patterns in the $S_{SafeSVs}$ set. Thus, we can remove it from the $\overline{S}_{ProbableAttack}$ set. Therefore, one false alarm is detected by this method and classified as a non-attack.

4. Any observation that remains in $\overline{S}_{ProbableAttack}$ will be copied in $\overline{S}_{Attack}$.

Table 3.1 Euclidean distances on toy example

Safe SVs	Probable attacks	ED	modified
[4.51 7.83]	[3.13 6.4]	1.987	No
[5.4 0.02]	[3.13 6.4]	6.772	No
[4.51 7.83]	[2.4 1.29]	6.872	No
[5.4 0.02]	[2.4 1.29]	3.258	No
[4.51 7.83]	[8.05 5.51]	4.232	No
[5.4 0.02]	[8.05 5.51]	6.096	No
[4.51 7.83]	[8.12 7.86]	3.61	No
[5.4 0.02]	[8.12 7.86]	8.298	No
[4.51 7.83]	[7.54 0.17]	8.238	No
[5.4 0.02]	[7.54 0.17]	2.145	No
[4.51 7.83]	[6.82 7.]	2.455	No
[5.4 0.02]	[6.82 7.]	7.123	No
[4.51 7.83]	[5.49 0.06]	7.832	No
[5.4 0.02]	[5.49 0.06]	0.099	Yes
[4.51 7.83]	[7.5 3.49]	5.27	No
[5.4 0.02]	[7.5 3.49]	4.056	No

Table 3.2 summarizes all the performance metrics, and it is possible to observe that there is an increase in terms of the F-measure and ROC score value in the one-class SVM algorithm. This toy example suggests that it is possible to decrease this rate with the help of SVs, outliers and similarity between vectors based on Euclidean distance.

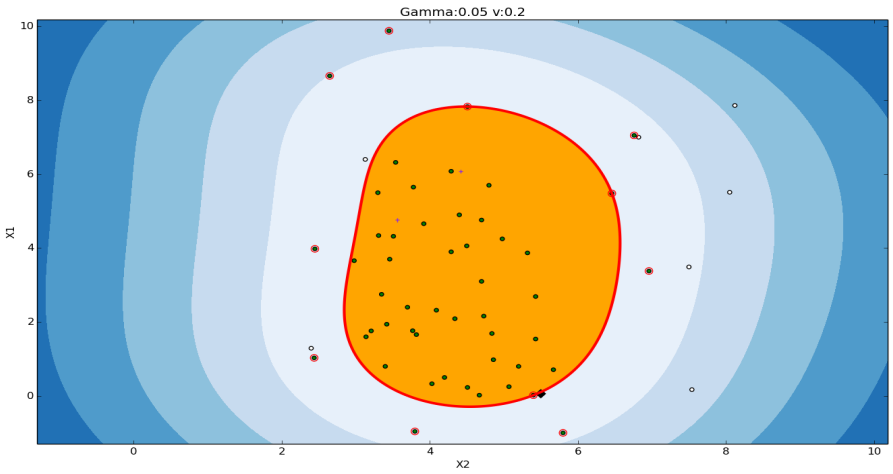

Figure 3.4 Scatterplot with decision boundary after reducing false alarms

Table 3.2 Performance of one-class SVM on toy example

Method	F-measure	ROC Score	FN	FP
One-class SVM	0.769	0.7	0	3
One-class SVM with FA reduction phase	0.833	0.8	0	2

3.4 Comparison with other methods

Our proposed approach has two advantages compared to other methods, namely, low computational time and minimal human dependency.

We use feature selection in the preprocessing phase of the data, as explained in section 4.2. When one algorithm uses fewer features, it has a lower training time compared to using all the features in model training (suggested in section 4.8).

Moreover, our method is not human dependent in the false alarm reduction phase, which means that the system automatically reduces this rate without human interaction. However, when evaluating anomalies or attacks detected by an intrusion detection system, there needs to be some human interaction for any attack observation.

CHAPTER 4 TEST

This chapter starts with a detailed explanation of the data and data preprocessing step and continues with how we test our proposed technique for reducing false alarms. For the testing phase, three different scenarios and three different one-class classification algorithms are investigated. Next, the results from testing the proposed method on the KDD99 database are briefly explained. Finally, we discuss the impact of sample size and dimensionality versus training time, FP reduction time and F-measure metric on artificial datasets.

4.1 Real Dataset

The data provided by *Groupe Access* are captured from hardware indicators every 30 seconds, and we refer to these data as the real dataset. The columns of these data are called attributes or features, and the rows of the data are called instances or observations. The data are in two different files: one file that consists of only non-attack observations and one file that contains attack observations in addition to non-attack patterns. We combined these two files, resulting in one file with a large amount of non-attack observations and a small amount of attack patterns. Then, we use 80% of the data for training, named *train file*, and the remainder of the data for testing, called *test file*. In general, in the domain of anomaly detection, the number of non-attack patterns is considerably higher than that of attack observations. In the provided data, the number of non-attack observations is 30295, whereas the number of attack observations is only 65. Note that the data are randomized before being divided to be able to make the assumption that the data are identically distributed.

The train file is used for parameter estimation and training the model; the test file is utilized for the testing phase. In this method, no labels are used in the training phase since this algorithm is unsupervised. However, we employ attack patterns for the model selection phase.

The given data have 9 different features captured from server logs. These features are hardware indicators, including nw_ in, nw_ out, CPU fan, CPU temp, CPU voltage, CPU usage, CPU load, available memory and free memory.

Feature nw_ in shows the number of input packets going to the server, and feature nw_ out depicts the number of output packets coming from the server. Fan is the cooling device on the computer systems, where the feature CPU fan shows the speed (how fast it is operating). CPU temp is the temperature of the CPU. CPU voltage is the amount of power that one

CPU consumes. CPU usage shows the fraction of CPU that is used for processing. When this value is high, there is a high usage of resources. Accordingly, we can conclude that attackers may be misusing resources to obtain some information. CPU load indicates the rate of processes that are waiting to be processed by the CPU. Free memory depicts the unused memory by the system, whereas available memory shows both the unused and cached memory.

Before choosing a method to train the model and initiate the detection phase, it is recommended that several techniques be applied to obtain an initial possible understanding of the data distribution and their correlations. In some cases, plotting the data results in an enormous amount of information and shows the relationships among the features. For example, one way to understand the distribution of the data is through a *histogram*, which groups the given data into ranges and plots them as bars. The taller the bar is, the more data instances that exist in that range, and this plot depicts the quantity of data instances in different ranges or groups. Through the histogram, we can determine whether the data have a normal distribution.

Figure 4.1 presents the histogram plot for all the features. As shown in this figure, there is no sign of a normal distribution in these plots since the bars depict no bell curve. In fact, the normal distribution should have more examples around the mean of the data, whereas in this plot, we do not observe such behavior.

The behavior of the features is shown via the line plot in figure 4.2, which shows the values of all the features. Here, the vertical lines show the values of each feature, and the horizontal line shows the number of observations. In this plot, only three features (nw_ out, CPU usage and CPU load) show peaks at the time of attack (shown by the result subplot).

4.2 Data Preprocessing

Data preprocessing, or data pretreatment, has a substantial impact on the results of anomaly detection methods, such as computation time, increasing accuracy and lowering the false alarm rate. Generally, preprocessing consists of four phases: *data creation*, *feature construction*, *feature selection* and *feature transformation* (J.Davis and J.Clark, 2011). Data creation consists of finding the appropriate label or class for the observations. The construction of new discriminate features from primary attributes is feature construction. Feature selection is the process of omitting irrelevant or non-significant features from the data and responding to the curse of dimensionality problem in high-dimensional databases. Feature transformation is the process of manipulating the values of features. In this thesis, our concern is on the last

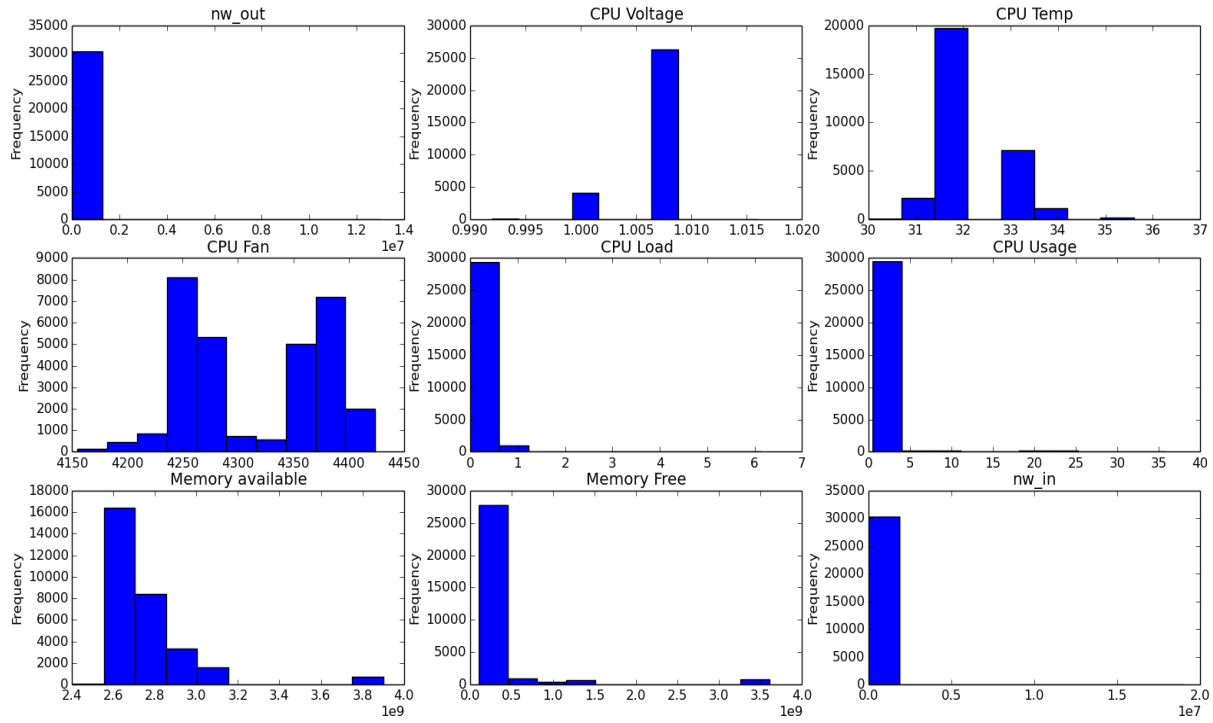

Figure 4.1 The histogram plot of all the features in the train file

two phases of data preprocessing. We describe two different data transformation methods: feature scaling and robust scaling.

Feature scaling is a normalization technique that aims to transform all the data values to a special range. Normalization techniques are included in the data transformation category because they modify the feature values. Due to the presence of different scales and ranges for each feature in the real data, it is recommended that scaled and preprocessed feature values be extracted to prevent the large numerical values from dominating the performance (J.Davis and J.Clark, 2011).

$$z_k = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{4.1}$$

where x_{min} shows the minimum value and x_{max} depicts the maximum value of each feature. Using function 4.1, it is possible to transform the values of the features into the range of $[0, 1]$.

Robust scaling starts by removing the median and continues by re-scaling the data to the *interquantile range*. This range shows the range between the first and third quartiles of the

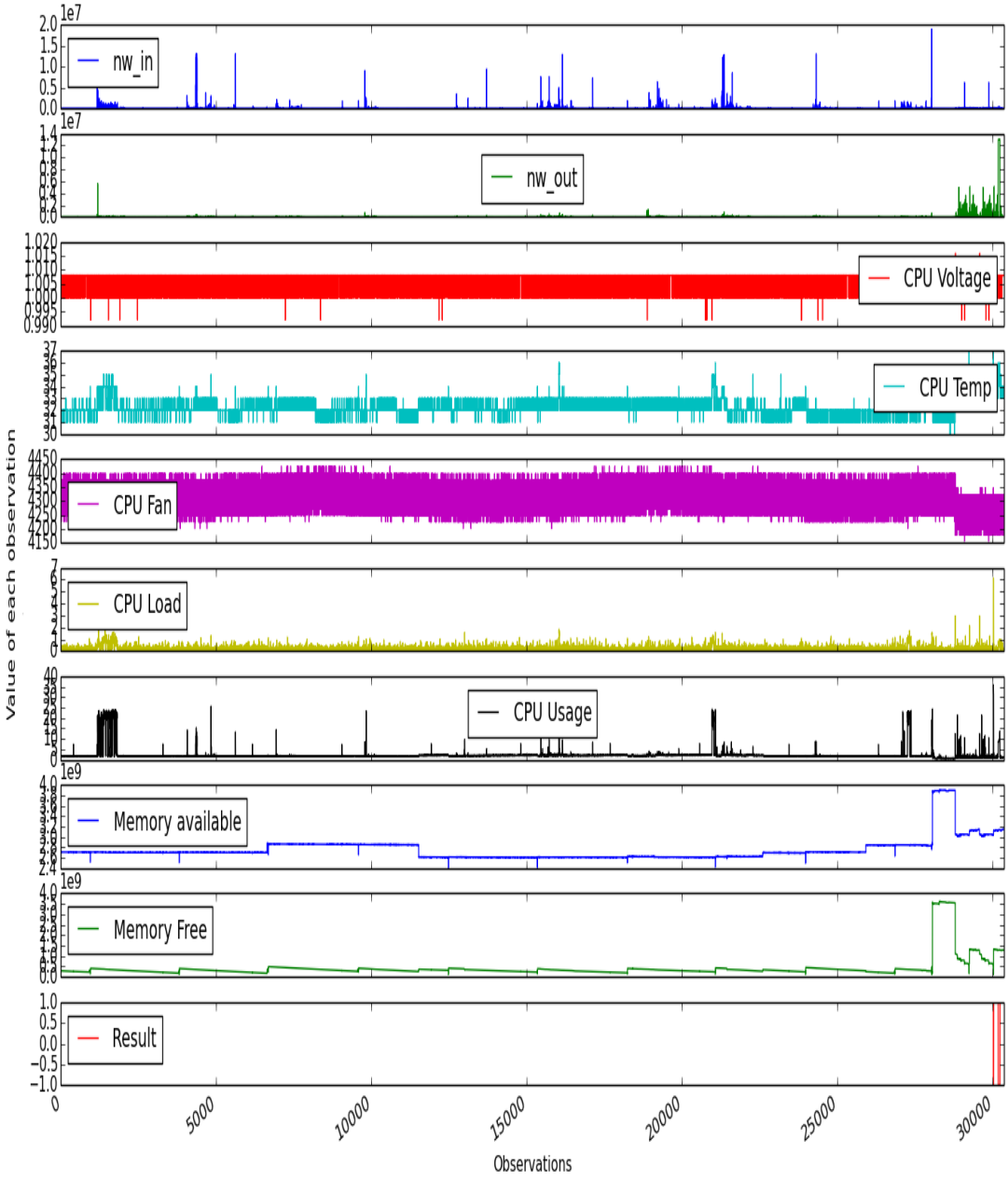

Figure 4.2 Line plot for all the features

data. Specifically, it calculates the difference between the third and first quartiles. This data transformation method is robust to outliers.

Although feature scaling is sensitive to outliers, we investigated its influence on one-class SVM. For this purpose, we proposed the following two approaches:

- We remove non-target examples from the training set and use the remaining examples for rescaling. Consequently, min and max values are drawn from target patterns, leading to not placing much importance on non-target examples. We define this approach as *feature scaling without the non-target patterns*.

- We remove non-target and outlier patterns from the training set and use the remainder for scaling the data. We define this approach as *feature scaling without non-target and outlier patterns*.

Moreover, we employ robust scaling, which is not sensitive to outliers. The results show that feature scaling achieves better accuracy and F-measure values compared to the robust scalar. All the results of the robust scalar are provided in appendix B. Consequently, we use feature scaling to $[0, 1]$ as the chosen feature transformation method in the remainder of this thesis.

Therefore, to preprocess the real data, we applied the following two steps:

- Feature selection: Feature selection or variable selection is a crucial preprocessing step in machine learning modeling, with the aim of removing redundant and irrelevant features from the data. Consequently, the predictive model becomes simpler and more efficient. Regularization or penalization methods such as Lasso regression (Tibshirani, 1996) have been shown to be effective for this purpose since they choose the most relevant set of features, and they are computationally efficient. Lasso regression is a frequently used technique that selects a subset of more important features. In greater detail, it forces the irrelevant features to have a coefficient of zero. Another method to perform feature selection is visually selecting features. Hence, according to figure 4.2, we select the most important features, leading to three features. This decision is based on the high variation in these three features during the time of attack. Additionally, we use Lasso penalization, leading to two features (CPU voltage and CPU load). The result of this feature selection leads to lower F-measure rate, as presented in appendix C.

- Data transformation: To prepare the data for training, it is necessary to scale the data. For this purpose, scaling the values in the range of $[0, 1]$ is employed to manipulate the data. The one-class SVM and given data led us to implement the pretreatment of the data since with great differences in the ranges of feature values, non-pretreatment results in a high number of false alarms. In fact, having some features with values that are greater than those of other feature leads these features to dominate the remaining features.

4.3 Model Selection

In general, in model selection between classifiers, the selected classifier should be flexible enough to depict the data well, and it should be simple enough to avoid overfitting, i.e., having a low training error and high generalization error (Tax, 2001). Moreover, the primary objective in model fitting is to find a model that can generalize well on the unseen observations.

Recall that one-class SVM has a user-defined parameter called ν. Thus, we need to tune this magic parameter before training the model. Moreover, we need to choose a proper kernel. If we choose the RBF kernel, then we need to tune another parameter called γ. Consequently, when we use the RBF kernel, we need to tune both γ and ν. Thus, the model selection in this case is to choose a proper value for ν to control the number of outliers and SVs, as well as tuning the kernel parameter. To tune these parameters, we use K-fold cross-validation. First, we select 20 percent of the training patterns for the testing phase (unseen observations). Then, we perform 5-Fold cross-validation on the remaining 80 percent to tune the hyper-parameters. In more detail, we divide the data into 5 sections, and then we use 4 sections to train and one section as the validation set. This process should be performed 5 times. Once the test is performed (the average error on the validation set is calculated), the validation set is reinserted into the training database, and another part is used for validation. Once the five steps are executed, the total average error is obtained from the average errors obtained at each step. Based on this average error, we can obtain the best values for these magic parameters. Here, we use the F-measure metric as the error.

In the case of the RBF kernel, we test different possibilities of the parameter values on the real data, as follows:

- ν (0.001,0.002,0.,...,0.02)
- γ values between 0.05 and 0.95 with a step of 0.1

Based on these various possible values, we have 200 possible pairs of values for these two hyper-parameters. Recall that when applying one-class SVM for the purpose of anomaly detection, we tend to choose small values for parameter ν (Schölkopf et al., 2001). In fact, in outlier detection, we always consider the number of outliers to be considerably smaller than the number of non-attack observations. Since we randomize the data, we perform the model selection procedure 10 times to obtain consistent values for the parameters. Table 4.1 summarizes the average F-measure, average FN, average FP, average AUC, the best value for parameter ν and the chosen value for parameter γ for each of the 10 runs. Based on this

result, we select a value of 0.004 for parameter ν and a value of 0.15 for parameter γ. This selection is based on the majority of this combination.

Table 4.1 Average performance of one-class SVM in 10 different runs

	AvgF-measure	avgFN	avgFP	avgAUC	ν	γ
Run # 1	0.669	2	8	0.913	0.003	0.25
Run # 2	0.674	0	11	0.97	0.004	0.15
Run # 3	0.703	0	11	0.99	0.004	0.15
Run # 4	0.672	2	8	0.913	0.003	0.15
Run # 5	0.663	0	11	0.97	0.004	0.15
Run # 6	0.671	0	12	0.98	0.004	0.15
Run # 7	0.659	0	12	0.97	0.004	0.15
Run # 8	0.659	0	12	0.97	0.004	0.05
Run # 9	0.676	0	12	0.96	0.004	0.25
Run # 10	0.639	1	12	0.961	0.004	0.05

According to section 2.5.3, we saw that small values for parameter ν and γ result in better classifiers (smooth and enclosed). This behavior suggests that large values for these parameters have a greater FP and FN rates. On the other hand, parameter ν shows the fraction of outliers (65 patterns) with respect to target examples (30295 patterns). Based on these reasons, we only investigated small values for these two parameters.

4.4 Visualization Methods

This section presents brief explanations of the ROC graph and confusion matrix plot.

4.4.1 ROC Graph

The ROC graph is a method to present the performance level of various classification methods (Fawcett, 2006). It is used to denote the trade-off between FP rate and TP rate on ROC space. The curve is two-dimensional and constructed based on the FP rate and TP rate to demonstrate the relative trade-off between them.

Figure 4.3 shows the basic ROC graph with two different classifiers. Point A illustrates perfect classification with a TP rate of one and a FP rate of zero. Point B, with a TP rate of

0.8 and FP rate of 0.4, exhibits a lower accuracy. Typically, if one point is located northwest of another point in the graph, we can conclude that the classifier performed better compared to a point closer to the $y = x$ line. The diagonal line $y = x$ shows a random guess where the algorithm produces a negative result in half of the cases, and in the other half of the cases, it produces a positive result.

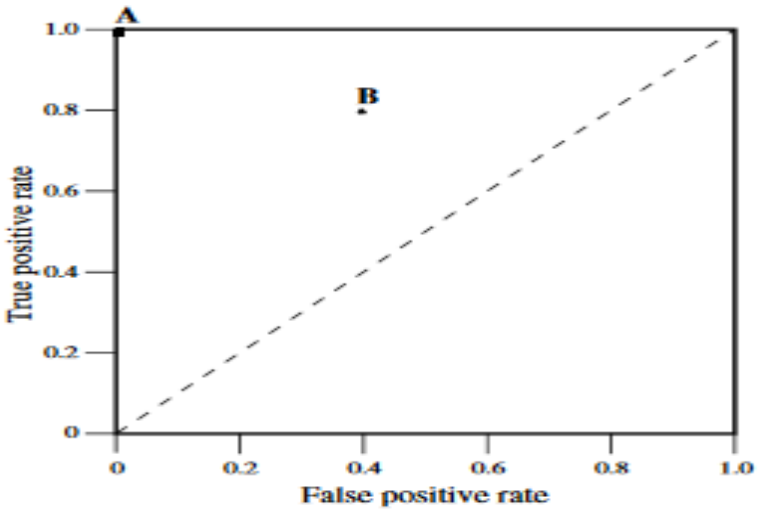

Figure 4.3 Basic ROC graph showing two classifiers

Note that the classifier that we trained in this thesis is a discrete classifier, i.e., it only produces label for each observation. In this case, we can only have points in the ROC graph. However, it is possible to draw a line from the origin to these points and a line from these points to the $(1, 1)$ point in the ROC space. Therefore, we can have a solid line for each of the classifiers. This will enable us to also compute the AUC value and compare various classifiers.

4.4.2 Confusion Matrix Plot

The confusion matrix plot is a technique for demonstrating the performance of the classification method, as shown in figure 4.4.

In the basic confusion matrix plot, the non-diagonal values of the matrix depict the misclassified observations, and the diagonal values represent the number of instances that are classified correctly. In addition, the number of misclassified and the number of successfully classified instances are depicted by color, as shown by the ruler on the right side of the figure. If the non-diagonal instances of the confusion matrix are white, then we can conclude that the algorithm failed in its classification. However, if the diagonal instances are dark blue,

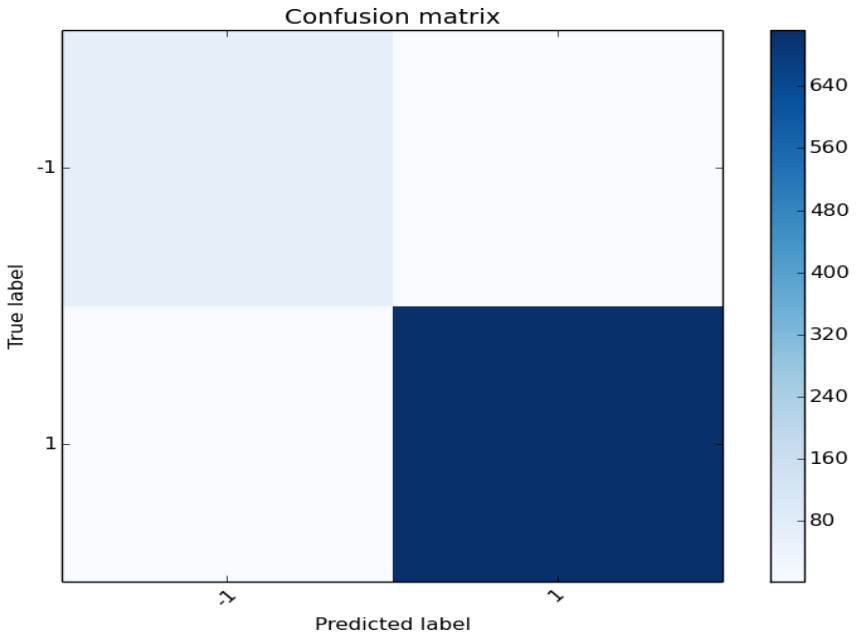

Figure 4.4 Basic confusion matrix plot

then it demonstrates that the model was successful in the classification.

4.5 Empirical Result

Three different scenarios based on one-class SVM were tested, showing the higher performance of the proposed method in decreasing the false alarm rate. The first is a scenario that only uses a one-class SVM to detect anomalies. The second scenario shows the proposed false alarm reduction method. The third scenario has the same steps of scenario 2 but without outliers. According to the experimental results, the second approach achieved the same ε_I compared to the two other scenarios, but with a considerably lower amount of ε_{II}. The details of these scenarios are described in the following sections.

4.5.1 Scenario 1

In scenario 1, a one-class SVM is tested for anomaly detection. For preprocessing data, we use feature scaling without non-target patterns to remove the impact of them on the results. Additionally, we employ the calculated parameters from section 4.3 to train the model, and we use the test observations to evaluate how well the model can generalize on the unseen observations. Figure 4.5 shows the steps of scenario 1. The name and the method or function used for each step are shown in this figure. For instance, feature selection is one step that is

performed visually.

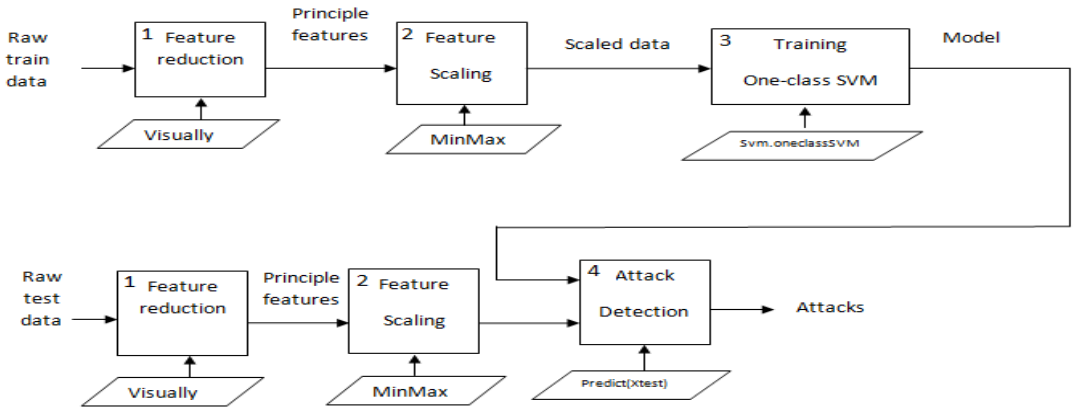

Figure 4.5 Project structure of one-class SVM (scenario 1)

The following steps were executed to implement this scenario:

1. Feature reduction: visually
2. Data transformation: feature scaling without non-target patterns
3. Training the model and calculating the frontier: one-class SVM with selected parameters ($\nu = 0.004$ and $\gamma = 0.05$)
4. Detecting attacks: in this step, the test file, which contains the attack observations, was used for the classification, and any observation that falls outside the calculated frontier is classified as an attack. The remainder of the unseen observations that are accepted by the classifier are classified as non-attacks.

Figure 4.6 presents the 2D calculated decision boundary using one-class SVM on three different plots. Since this is a 2D representation of the real decision boundary, we can only show 2 features in each subplot, leading to three different plots. In all of these subplots, the frontier is the same, but the scatter points are varied. It is possible to observe that the points outside the orange circle are outlier observations or anomalies, and the points inside the circle are the target instances. Note that the scatterplot is a 2D representation rather than real dimensions. Meanwhile, the plotted decision boundary itself is constructed using all the features of the training data. This fact justifies why there are some target points outside of the circle in all these subplots.

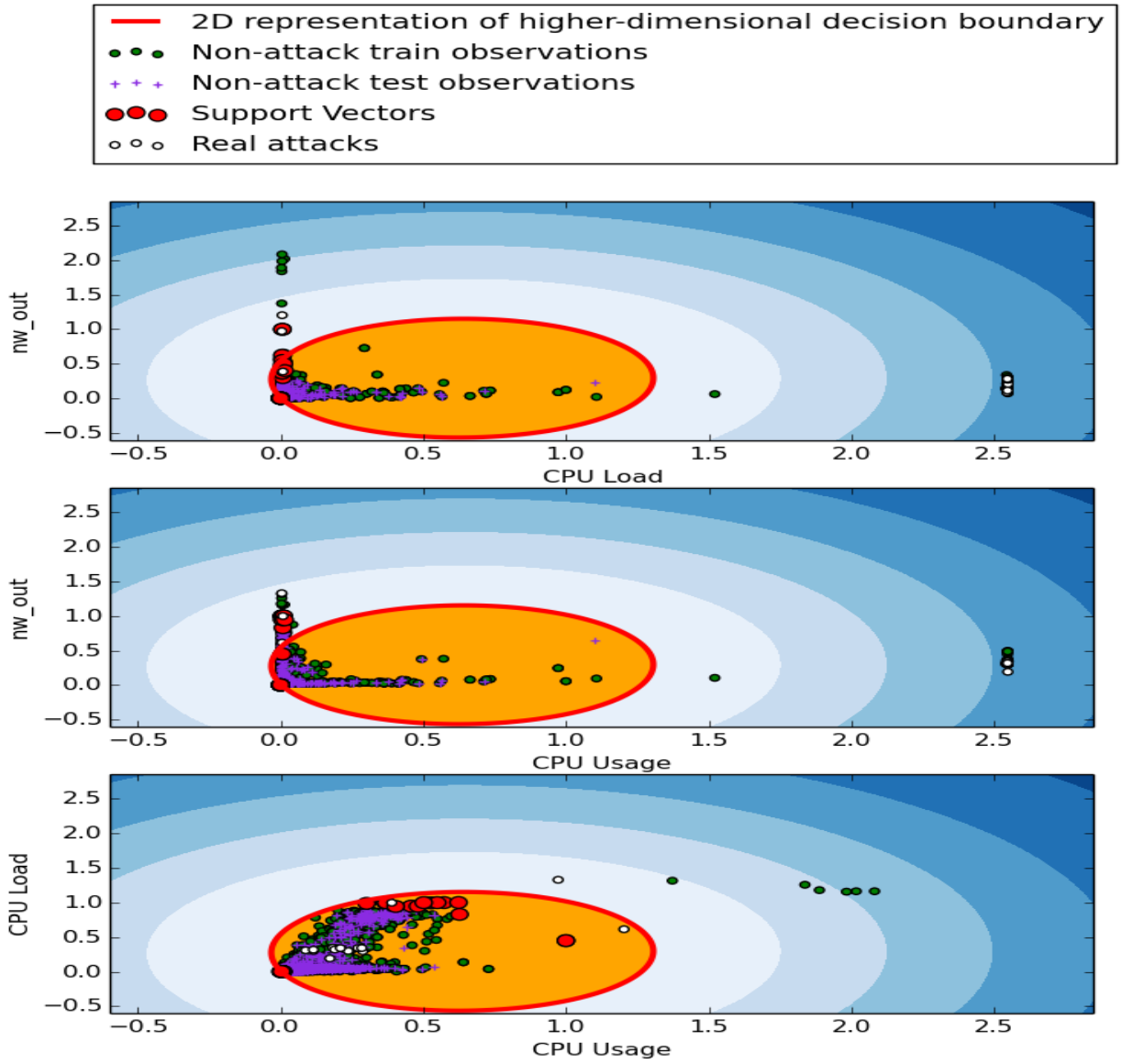

Figure 4.6 Scatterplot with decision boundary (scenario 1)

4.5.2 Scenario 2

In this scenario, we investigate the influence of the false alarm reduction method. For the preprocessing phase, we use feature scaling without non-target patterns. For training the model, we employ one-class SVM, and finally, we add the false alarm reduction phase to evaluate its influence on ε_I and ε_{II}. With the help of outliers and SVs, it is possible to decrease the amount of false alarms such that they have no impact on the detection of true

alarms. In this phase, probable attacks were detected by testing the test file, but for deciding whether they are real attacks, we compared them with SVs minus non-target observations known as safe SVs (cf. section 3.1). If one probable attack has the same pattern as one safe SV, then we can conclude that it is a false alarm. Consequently, only those patterns that are not the same as any safe SV pattern and any non-attack pattern are reported as true attacks. Hence, this scenario is implemented as follows:

1. Feature selection: visually
2. Data transformation: feature scaling to $[0, 1]$ without non-target patterns
3. Training the model and calculating the frontier: one-class SVM with selected parameters ($\nu = 0.004$ and $\gamma = 0.05$)
4. Tuning parameter T with 5-fold cross-validation: we choose the best value for this parameter by training the model with different possible values ($T = \{0.01, 1\}$ with step 0.01) based on the AUC value (0.9615). For these real data, there are multiple best choices, and based on the assumption, we choose the smallest possible value. Figure 4.7 depicts the relation between AUC and parameter T.
5. Detecting probable attacks: in this step, the test file, which contains the attack observations, is used for the classification, and any observation that falls outside the calculated frontier is classified as a probable attack.
6. Reduction of false alarms and detection of attacks: calculating safe SVs set, comparing the probable attacks with safe SVs based on Euclidean distance, and concluding real attacks.

Figure 4.8 presents the steps of the proposed scenario.

Figure 4.9 shows the scatterplot with decision boundary for scenario 2. The black diamonds are the detected false positive observations.

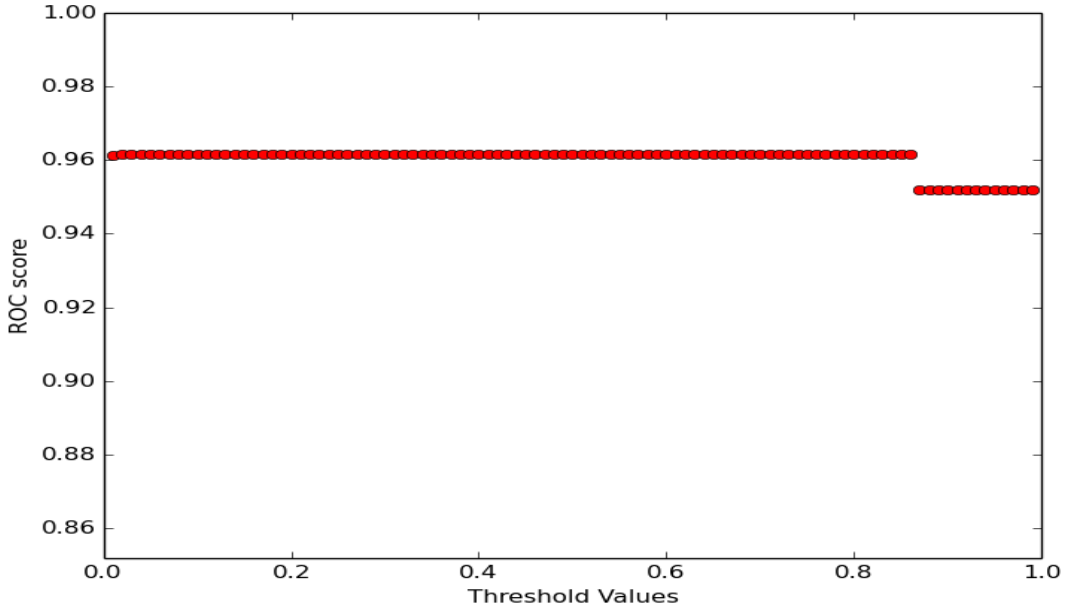

Figure 4.7 AUC versus parameter T (scenario 2)

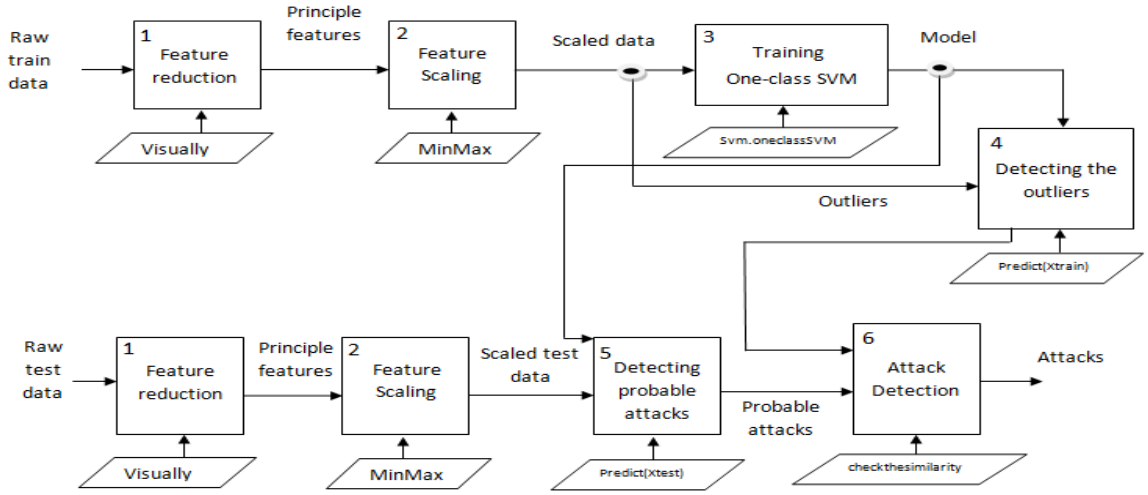

Figure 4.8 Project structure of one-class SVM (scenario 2)

4.5.3 Scenario 3

In scenario 3, we assume that 1 percent of the observations are outliers. These outliers are detected by one-class SVM ($\nu = 0.01$ and $\gamma = 0.15$). We transform the feature values based on training data without non-attack and outlier patterns. Once the outliers are detected,

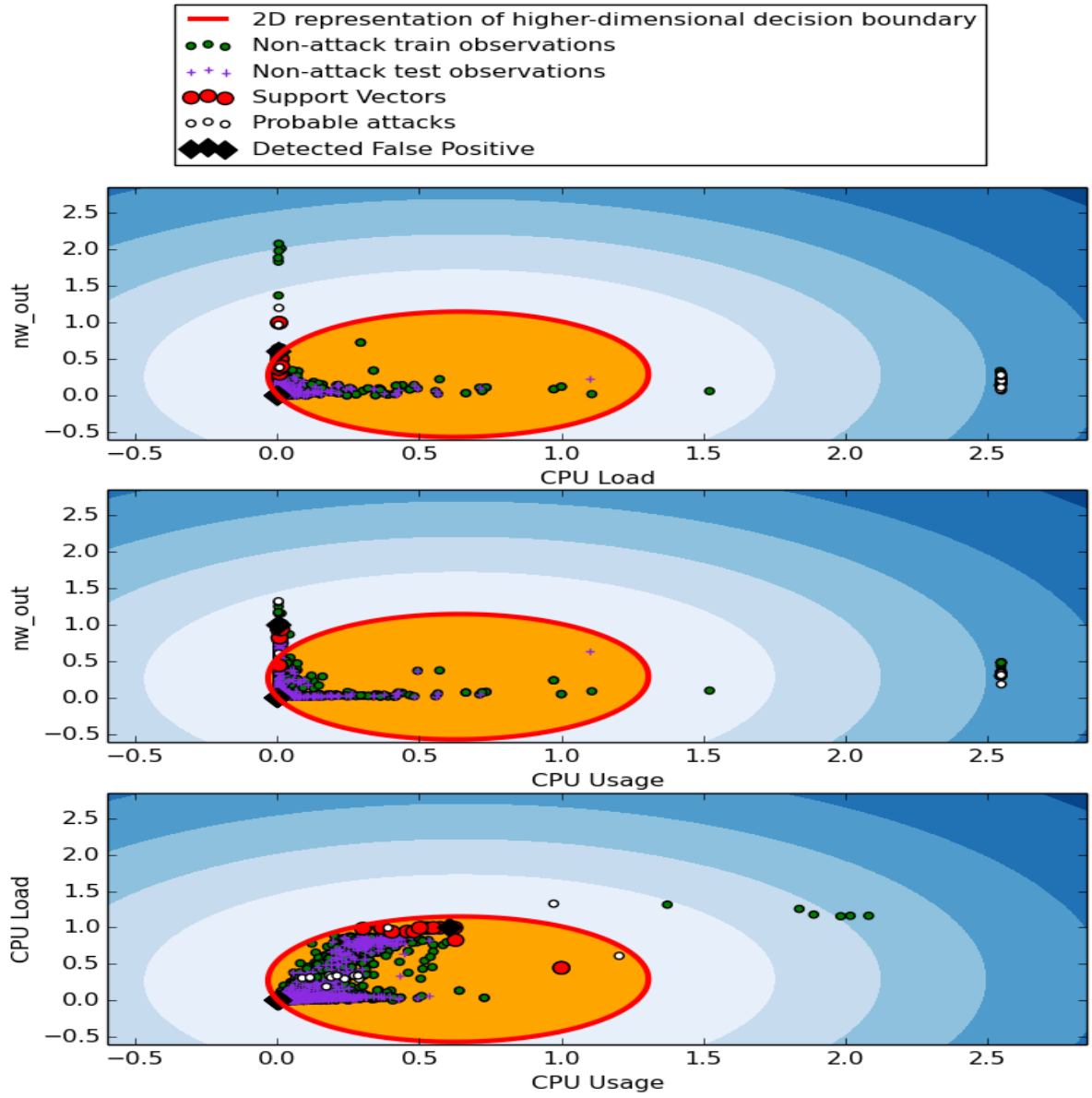

Figure 4.9 Scatterplot with decision boundary (scenario 2)

we can remove them from the training set. In fact, this scenario is proposed to evaluate the effect of removing outliers on the overall performance. Hence, the following hypothesis should be verified:

- H0d: Removing outliers from the training observations leads to lower F-measure values (increasing ε_{II}).

Figure 4.10 presents the different steps of the third scenario.

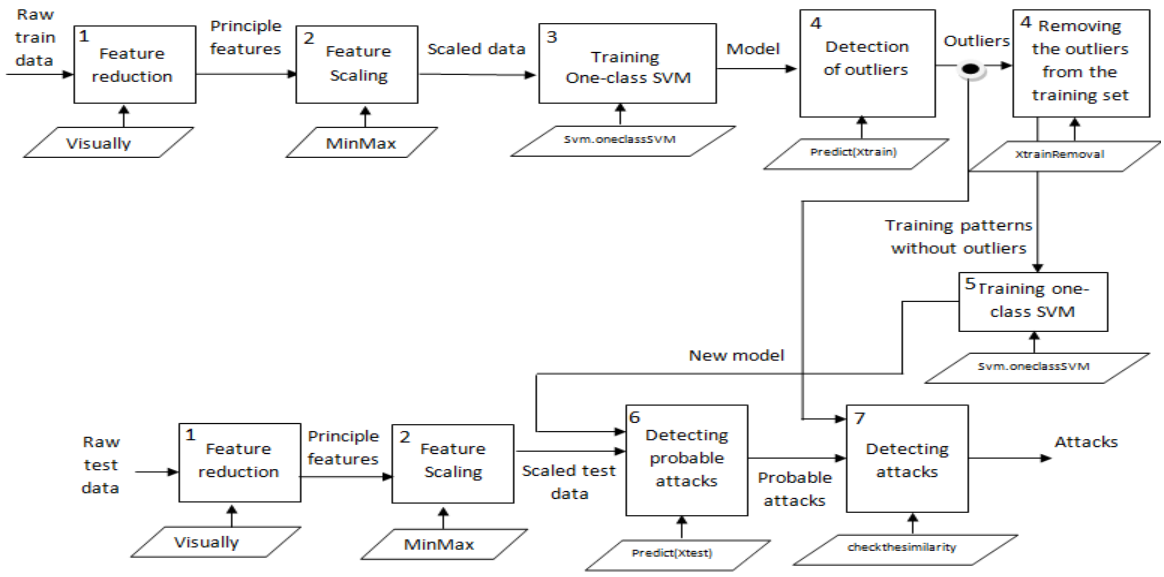

Figure 4.10 Project structure of the one-class SVM (scenario 3)

1. Feature selection: visually

2. Data transformation: feature scaling to $[0, 1]$ without non-target and outlier patterns

3. Training the model and calculating the frontier: one-class SVM with selected parameters ($\nu = 0.01$ and $\gamma = 0.15$)

4. Detect and Remove outliers: we remove outliers based on the calculated frontier

5. Tuning parameter T: we choose the best value for this parameter by training the model with different possible values as in scenario 2 and selecting the best value for parameter T based on the AUC (0.885). Figure 4.11 depicts the relation between the AUC and parameter T.

6. Training the model and calculating the frontier: one-class SVM with selected parameters ($\nu = 0.003$ and $\gamma = 0.05$)

7. Detecting probable attacks: in this step, the test file, which contains the attack observations, is used for the classification, and any observation that falls outside the calculated frontier is classified as a probable attack.

8. Reduction of false alarms and detection of attacks: calculating safe SVs set, testing unseen observations on the trained model, and finding probable attacks, comparing these probable attacks with safe SVs, and concluding real attacks.

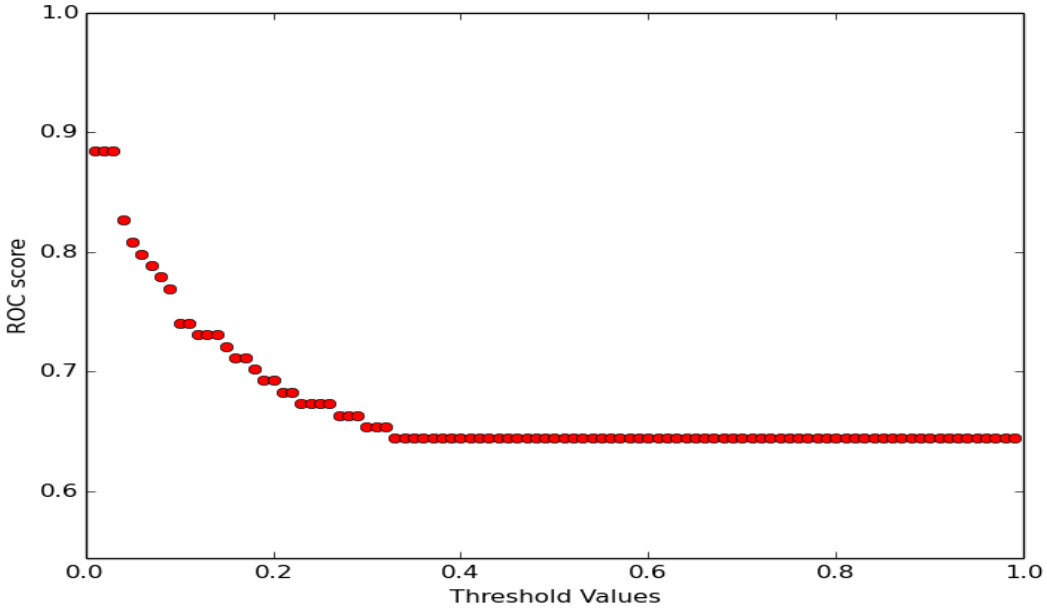

Figure 4.11 AUC versus parameter T (scenario 3)

Note that in this scenario, the values of the hyper-parameters are tuned again since we remove outliers with the upper bound of 1 percent. Based on figure 4.12, which shows the scatterplot with the decision boundary, we can observe that the decision boundary has changed for all three subplots. The non-attack area becomes smaller compared with the subplots in figure 4.6. This change is acceptable since this scenario has a different set of training patterns compared to the two other scenarios, which is why the safe zone has diminished, i.e., the zone where non-attack patterns should be located. Hence, through a change in the feature variances, the resulting decision function is modified.

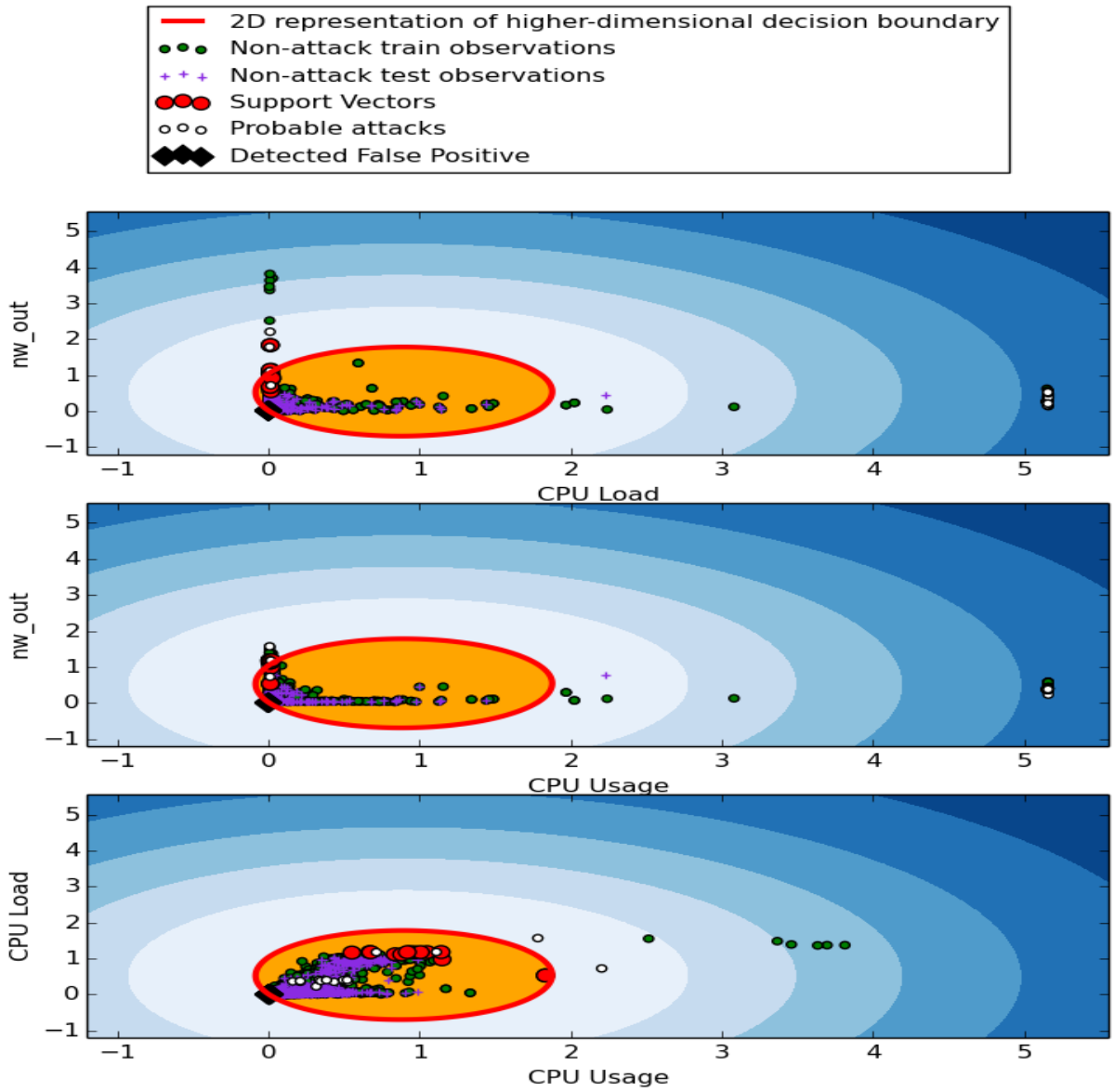

Figure 4.12 Scatterplot with decision boundary (scenario 3)

4.5.4 Comparison of the three scenarios

According to the results presented in the previous sections, the most promising scenario is the second one. Table 4.2 summarizes the performance comparison among the three proposed scenarios. The value of recall is constant since the number of misclassified true attacks remains the same in all three scenarios. This constant number of detected true attacks shows that adding the false alarm reduction does not affect ε_I. The result also reveals the influence

of reducing the false alarms via the metrics of AUC, precision and F-measure. The highest performance is obtained through the second scenario, with a value of 0.963 for the F-measure metric. This shows a 0.344 better performance in terms of F-measure compared to scenario 1 and 0.522 better compared to scenario 3. In addition, scenario 1 exhibits higher F-measure performance compared to scenario 3. This result suggests that removing outliers deteriorates ε_{II}. Note that in both scenario 2 and scenario 3, we use the false alarm reduction step; however, in scenario 3 we remove the outliers with the upper bound of 1 percent. Moreover, this result verifies assumption H0d.

Table 4.2 Performance of one-class SVM in three scenarios

Scenario	AUC	Precision	Recall	F-measure	avg computation time (s)
Scenario 1	0.999	0.448	1	0.619	0.306
Scenario 2	0.999	0.928	1	0.963	0.336
Scenario 3	0.997	0.867	1	0.441	0.58

The computation time for the training of 28488 (80 percent) training patterns is 0.274 seconds, and the computation time for the prediction of 6079 (20 percent) test patterns is 0.032 seconds. Hence, the computation time for scenario 1 is 0.306 seconds. Moreover, the computation time for the false alarm reduction phase is 0.03 seconds. Therefore, the total computation time for scenario 2 is 0.336 seconds. The computation time for scenario 3 is $0.274 + 0.306 = 0.58$ (the computation time here is in the average of 25 runs).

The confusion matrix plot is a technique for visualizing the misclassification and true classification rates. The confusion matrices plotted in figures 4.13 to 4.15 present the values of FN, TP, TN and FP for all the three scenarios.

In figure 4.16, another technique for comparing different scenario performances, namely, ROC graph, is presented. According to this figure, all three curves show the same TP rate, while the red line representing scenario 2 demonstrates better performance in terms of FP rate. This result suggests that the proposed second scenario with the false alarm reduction phase is better in terms of FP rate than the two other scenarios. Recall that in this real dataset, the number of non-attack observations is considerably higher compared to attack patterns; hence, the differences between ROC score values are very small. Consequently, we assume that there are only 400 non-attack observations. Based on this assumption, we can observe that the differences become larger and consequently more visible.

According to plots 4.6, 4.9 and 4.12, we observe a relationship between two features CPU load and CPU usage. One approach is to remove one of them and calculate the performance.

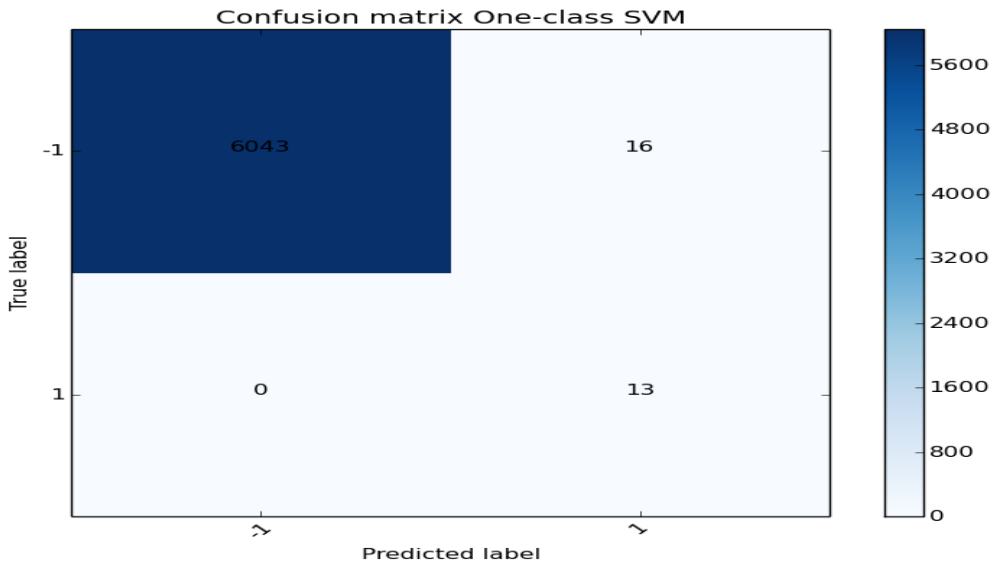

Figure 4.13 Confusion matrix plot for one-class SVM using the data without outliers (scenario 1)

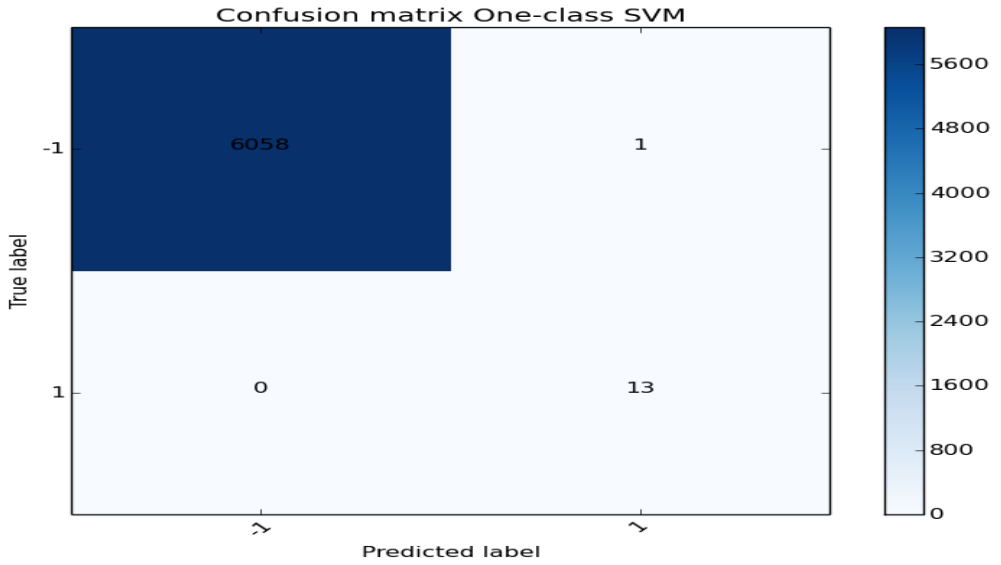

Figure 4.14 Confusion matrix plot for one-class SVM using the data without outliers (scenario 2)

However, according to annex B, removing one of these features results in poor F-measure value. Although we give more weights to almost same information, we gain a better result. Hence, we keep both features to achieve a better value of F-measure. Moreover, in all of these plots, although the decision boundary is close to the origin, it never passes the exact point of (0,0).

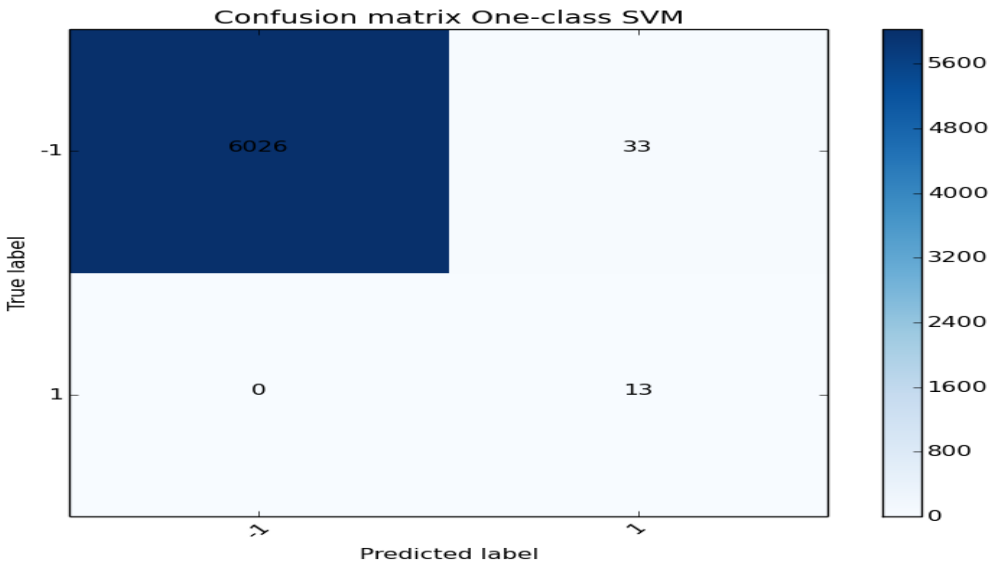

Figure 4.15 Confusion matrix plot for one-class SVM using the data without outliers (scenario 3)

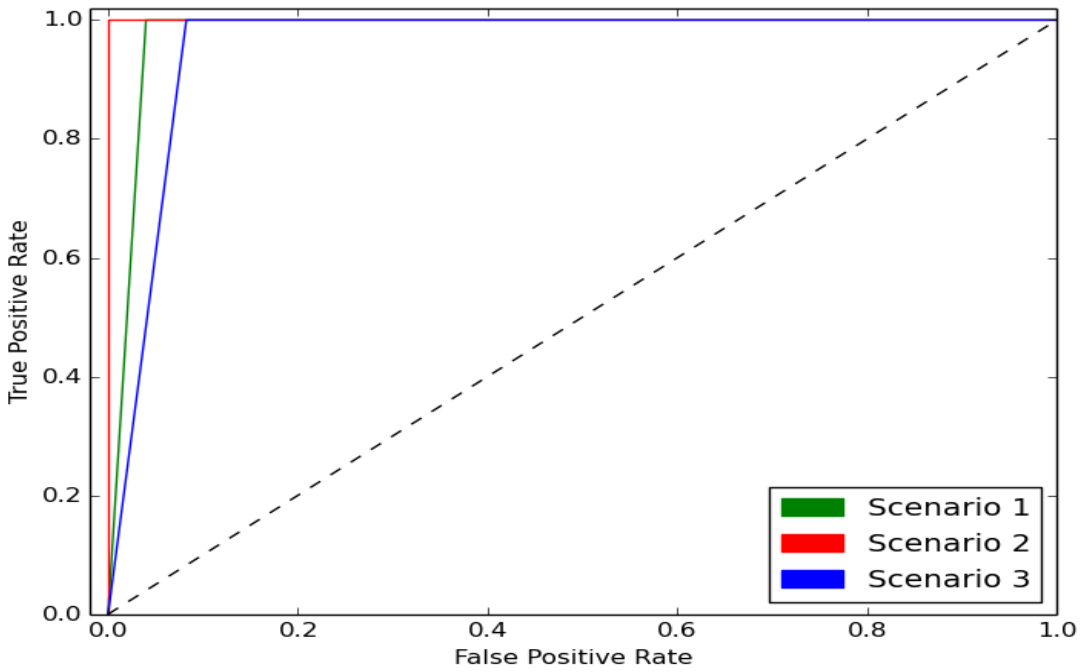

Figure 4.16 ROC graph for one-class SVM (all three scenarios)

These experiments are performed using the Ubuntu 14.04 64*bit* operating system running on a system with an $IntelCore^{TM}i5CPU2.40GHz$ with $4GB\ RAM$. Additionally, we use Python 2.7 scripting language to implement these algorithms. Moreover, for training one-class SVM, we use a library called scikit-learn developed by Pedregosa et al. (2011).

4.6 Empirical results on other algorithms

In this section, we compare our proposed scenarios with three different one-class classification algorithms on the real dataset. These algorithms are MoG density estimation, k-means data description and Parzen density. To implement these algorithms, we use a toolbox in MATLAB called *dd_tools* provided by Tax (2015).

Note that we use 5-fold cross-validation for all of these algorithms to tune their hyper-parameters. In Parzen density estimation, the rate of outliers was tuned to 0; in MoG density, the rate of outliers was tuned to a value of 0.001 with N_{MoG} equal to 2; and in k-means data description, the rate of outliers was tuned to 0 with a value of $k = 3$. Note that we only adjust the magic parameters and the free parameters are adjusted automatically by the algorithm.

Table 4.3 summarizes the different algorithms on the real dataset, suggesting that the second scenario achieved a comparable result to k-means data description. The third best result is for MoG density estimation with an F-measure value of 0.839. From all the methods, Parzen density depicts the lowest value of F-measure (0.341). As suggested in Tax (2001) thesis, Parzen density is a very weak algorithm since it does not have any magic parameter to adjust, and if the data does not represent the characteristics of the data, it fails to find a good classifier that can generalize well.

k-means shows the lowest training time comparing all the other methods. In general, all of these methods have lower training time comparing one-class SVM scenarios because we make assumptions regarding the data distribution. It should be noted that these three techniques are implemented in Matlab, whereas the one-class scenarios are implemented in Python.

It should be emphasized that all of these methods are successful in detecting real attacks since the value of FN is zero. Furthermore, there are almost 6000 observations in the test set that only a small number of them are categorized mistakenly in outlier class. In other words, although the F-measure value for Parzen density estimation is low, this does not show that it completely fails. This result suggests that comparing other methods on this dataset it resulted in the highest FP rate.

Figure 4.17 shows ROC curves for these three algorithms and three scenarios. The worst

Table 4.3 Comparison of one-class classification methods

	Parzen	MoG	k-means	Scenario 1	Scenario 2	Scenario 3
F-measure	0.341	0.839	1	0.619	0.963	0.441
FP	50	5	0	16	1	33
FN	0	0	0	0	0	0
avg training time (s)	0.032	0.125	0.081	0.306	0.336	0.58

one-class classifier is Parzen density and the best frontier is scenario 2.

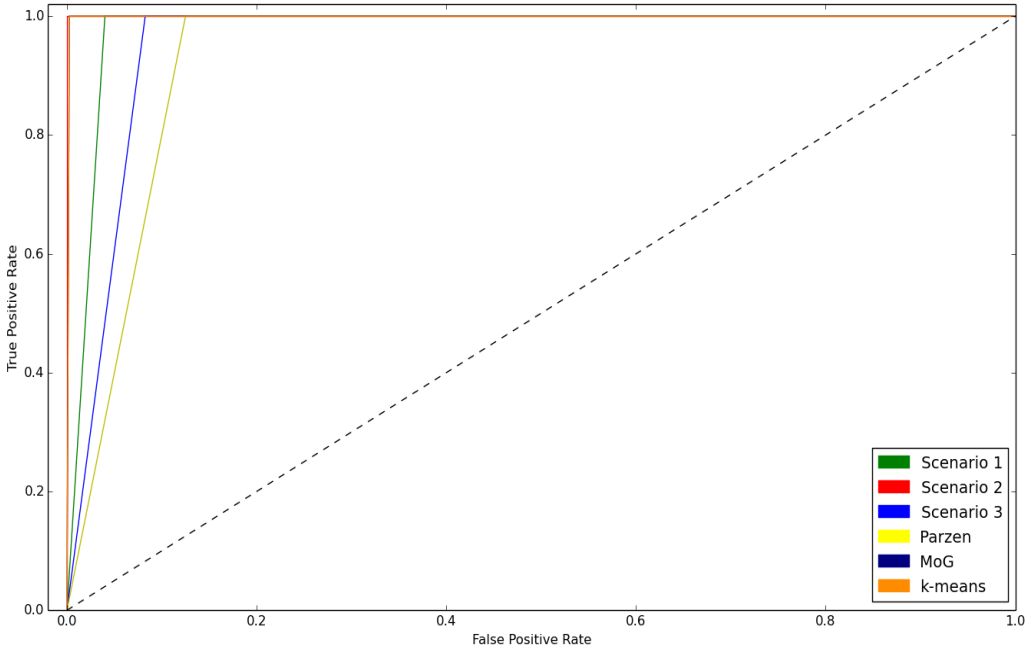

Figure 4.17 ROC curves for various one-class classification algorithms and three scenarios

4.7 Empirical result on KDD99 dataset

KDD99 is a dataset obtained from TCP dump data by MIT Lincoln Labs in 1998. This dataset consists of more than five million observations and 39 different features (Goeschel, 2016). The values of these features are either categorical or numerical. We use 30000 target observations and 3502 outlier patterns. Based on the result of 5-fold cross-validation, we

compute the value of $\nu = 0.05$ and the value of $\gamma = 0.1$. For the feature selection step, we use the Lasso algorithm, leading to reducing the number of features to 6. The results of the first and second scenarios (averaged in 25 runs) are reported in table 4.4, suggesting that the proposed false alarm reduction step has reduced the FP rate (scenario 2).

Table 4.4 Experimental results of two scenarios on the KDD99 dataset

	avgF-measure	avgAUC	avgFP	avgFN
Scenario 1	0.612	0.805	365	231
Scenario 2	0.679	0.818	212	231

4.8 Impact of the sample size

To demonstrate the impact of sample size and dimensionality versus F-measure, FP reduction time and training time, we created six artificial datasets with various sizes (100, 500, and 1000 training patterns) and different dimensionalities (2D and 10D) that are normally distributed. These artificial datasets have a mean of zero and variance of 10. Moreover, we consider five outliers in all six of these datasets.

The experimental results in table 4.5 suggest that by increasing the number of observations, both the training time and FP reduction time increase. This behavior is the same for increasing the dimensionality. Moreover, having the same number of observations with increasing dimensions decreases the F-measure value. This result occurs because if we increase dimensions, more training points are needed to achieve a good decision boundary, as suggested in Tax (2001). However, with the same dimensions, if we increase the sample size from 500 to 1000, the F-measure value decreases, but if we increase the sample size from 100 to 500, the F-measure value decreases.

Table 4.5 Experimental results for various sample sizes

Sample size	Dimensions	Training time(s)	F-measure	FP reduction time(s)	ν
100	2	0.0007	0.657	0.0002	0.05
500	2	0.0011	0.682	0.0003	0.01
1000	2	0.0018	0.679	0.0005	0.005
100	10	0.0008	0.513	0.0005	0.05
500	10	0.0016	0.539	0.0007	0.01
1000	10	0.0025	0.405	0.0008	0.005

4.9 Discussion of empirical result

In this chapter, we discussed the test phase of our proposed method, and we compared it to a one-class SVM alone and a one-class SVM trained without the outliers. Through this comparison, we find that this method can decrease the false alarm rate. Under the same conditions, scenario 2 exhibited the best performance compared to the two other scenarios. Specifically, using the false alarm reduction method based on one-class SVM increases the F-measure by 0.522 compared to scenario 3. In addition, scenario 2 achieved 0.344 better performance in terms of F-measure compared to one-class SVM alone. This result suggests that our proposed false alarm reduction method can be used with this algorithm to reduce the FP rate and consequently results in better detection of true attacks.

Furthermore, we investigated the proposed method on the KDD99 dataset, which suggested that by adding the proposed false alarm reduction step to one-class SVM, there is a decrease in ε_{II}.

Finally, we suggested that by increasing the sample size and dimensionality, the training time and FP reduction time increase. Additionally, based on the experimental results, with the same sample sizes, the F-measure value decreases when the dimensionality increases.

Overall, based on the experiments on the real and KDD99 datasets, we can conclude that the proposed false alarm reduction method is effective in reducing ε_{II}. Moreover, one-class SVM with this step can obtain an F-measure value comparable to those obtained using the k-means data description and MoG density estimation methods.

CHAPTER 5 CONCLUSION

5.1 Advancement of knowledge

In the introduction, we addressed the problem of the high false alarm rate of anomaly-based intrusion detection systems and proposed using the outlier detection capability of the one-class SVM algorithm to solve this problem. We made a contribution using the outliers and SVs to reduce the number of false alarms provided by this algorithm. In this algorithm, the fraction of outliers can be adjusted by a user-defined parameter called ν. We tend to choose smaller values for this parameter for the task of outlier detection. In the proposed false alarm reduction method, we first need to calculate the set of SVs and outliers. Second, we remove outliers from the SV set, leading to a safe SV set. Third, we use the trained model on unseen observations to obtain a set of probable attacks. Fourth, we compare all of these probable attack observations with safe SV patterns based on Euclidean distance, and we remove any observations from the probable attack set if they have a smaller Euclidean distance value compared to parameter T. Finally, we copy all the observations remaining in the probable attack set as real attacks. This approach leads to a smaller value of FP rate. Parameter T plays the role of a threshold in this proposed method, and it can be tuned by K-fold cross-validation. We choose this value based on the ROC score metric.

In the process of training the one-class SVM algorithm, the RBF kernel was employed. Moreover, we selected features based on their variation in attack observations. In greater detail, we used feature scaling to $[0, 1]$ to transform the values of data. For scenarios 2 and 1, we removed the non-target observations to avoid the negative influence of these observations on feature scaling. Moreover, for scenario 3, we removed both outliers and non-target observations from the data prior to feature scaling. Then, we performed model selection using K-fold cross-validation, leading to proper values to train our model based on the one-class SVM algorithm.

We proposed three different scenarios, including one-class SVM alone, one-class SVM with false alarm reduction step and one-class SVM trained without outliers. In the first scenario, we implemented the one-class SVM. In the second scenario, we implemented one-class SVM, and we reduced the number of false alarms using the proposed false alarm reduction step. In the third scenario, before training the model, we removed the outliers from the training data, and then we trained the model, followed by reducing false alarms.

We also investigated the results of three different one-class classification algorithms, namely,

Parzen density estimator, MoG density and k-means data description, for comparison with our proposed scenarios. The results suggest that the second scenario, one-class SVM with false alarm reduction step, achieved a better F-measure value compared to MoG density estimation algorithm. The result obtained by k-means data description is 1 for F-measure metric. Moreover, the other one-class classification algorithm, Parzen density estimation, the lowest F-measure value comparing the other methods.

Subsequently, we investigated the experimental results of the three scenarios on the KDD99 dataset. The results again suggest that the proposed false alarm reduction method is effective.

Finally, we investigated the impact of sample size and dimensionality in one-class SVM, and we suggest that increasing dimensions decreases the F-measure value. This is because with higher dimensions, the sample size needed to model the one-class classifier becomes considerably larger as suggested in Tax (2001). Moreover, with increasing dimensionality, the training time and FP reduction time slightly increase.

5.2 Limits and constraints

The proposed method has the following limitations:

1. The provided data are real data obtained from servers, and we have a small amount of attack observations to perform model selection. In the case of data with no attack observations, the algorithm is not usable.

2. The proposed false alarm reduction method is based on SVs; therefore, we can only add it to techniques that are based on SVs.

3. The possible values for parameter T are related to the data transformation step, which shows the dependency of these values on the chosen data transformation technique. This means that we cannot provide a universal good value for this parameter; hence, this parameter should be tuned. However, the proposed method will still be efficient if we have a good value for this parameter.

4. The proposed method requires that we check all probable attacks with the calculated outliers and SVs. This shows that even true alarms should be checked using this technique, leading to having more computation time.

5. We have only tested the algorithm on numerical values. In the case of data with categorical values, the code should be modified.

5.3 Recommendations

For some of the aforementioned constraints in the previous section, we have some recommendations. For example, for limitation 1, we can create some artificial outliers as suggested by Tax (2001), leading to simulated attack observations. In fact, he considers a box around the data set that ensures all the training observations are in the box, and then he uniformly distributed the data (box-shaped). It is also possible to use structural approaches to generate outlier observations that can show the good characteristics of the data. Additionally, for limitation 3, one automatic approach is to calculate all the Euclidean distances and use statistical metrics to produce a good set of possible values for parameter T.

We observe that there is a relationship between two features in real data. One approach is to employ independent component analysis, extracting the underlying components in the data. These selected components are statistically independent and non-Gaussian. In this case, we can explore the effect of independent features on the performance of the false alarm reduction method.

It is possible to use an enhanced version of a one-class SVM such as the ETA version introduced by Amer et al. (2013) to investigate the influence of this method. Moreover, we have tested this method only in intrusion detection systems; however, we can evaluate its performance in other applications, such as health care, credit fraud detection or video surveillance.

One future avenue for research could be learning the patterns dynamically and using this to further reduce the false alarm rate. Consequently, we can build a real-time intrusion detection method where it can collect new patterns as either non-attack or attack to train the model in real-time and simultaneously detect real attacks.

REFERENCES

M. Amer, M. Goldstein, and S. Abdennadher, "Enhancing one-class support vector machines for unsupervised anomaly detection," in *Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description*. ACM, 2013, pp. 8–15.

J. P. Anderson, "Computer security threat monitoring and surveillance," James P. Anderson Company, Fort Washington, Pennsylvania, Tech. Rep., 1980.

D. P. Bertsekas, *Nonlinear programming*. Athena scientific Belmont, 1999.

C. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

C. M. Bishop, *Neural networks for pattern recognition*. Oxford university press, 1995.

V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

N. Devarakonda, Srinivasulu, V. Kumari, and A. Govardhan, "Intrusion detection system using bayesian network and hidden markov model," *Procedia Technology*, vol. 4, pp. 506–514, 2012.

R. O. Duda, P. E. Hart, D. G. Stork *et al.*, *Pattern classification*. Wiley New York, 1973, vol. 2.

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

K. Goeschel, "Reducing false positives in intrusion detection systems using data-mining techniques utilizing support vector machines, decision trees, and naive bayes for off-line analysis," in *SoutheastCon, 2016*. IEEE, 2016, pp. 1–6.

A. H. hassan, S. Lambert-lacroix, and F. Pasqualini, "Real-time fault detection in semiconductor using one-class support vector machines," *International Journal of computer theory and engineering*, vol. 7, no. 3, p. 191, 2015.

T. Hastie, R. Tibshirani, G. James, and D. Witten, *The Elements of Statistical Learning*. Springer, 2013, vol. 6.

J. J.Davis and A. J.Clark, "Data preprocessing of anomaly based network intrusion detection: A review," *Computer and Security*, vol. 30, no. 6, pp. 353–375, 2011.

S. Juma, Z. Muda, and W. Yassin, "Reducing false alarm using hybrid intrusion detection based on x-means clustering and random forest classification," *Journal of Theoretical & Applied Information Technology*, vol. 68, no. 2, 2014.

S. S. Khan and M. G. Madden, "One-class classification: taxonomy of study and review of techniques," *The Knowledge Engineering Review*, vol. 29, no. 03, pp. 345–374, 2014.

A. D. Landress, "A hybrid approach to reducing the false positive rate in unsupervised machine learning intrusion detection," in *SoutheastCon, 2016*. IEEE, 2016, pp. 1–6.

K.-L. Li, H.-K. Huang, S.-F. Tian, and W. Xu, "Improving one-class svm for anomaly detection," in *Machine Learning and Cybernetics, 2003 International Conference on*, vol. 5. IEEE, 2003, pp. 3077–3081.

W. Li, W. Meng, X. Luo, and L. F. Kwok, "Mvpsys: Toward practical multi-view based false alarm reduction system in network intrusion detection," *Computers & Security*, vol. 60, pp. 177–192, 2016.

S. M.Erfani, S. Rajasegarar, S. Karunasekera, and C. Leckie, "High-dimensional and large-scale anomaly detection using a linear one-class svm with deep learning," *Pattern Recognition*, vol. 58, pp. 121–134, 2016.

A. Mokarian, A. Faraahi, and A. G. Delavar, "False positives reduction techniques in intrusion detection systems-a review," *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 13, no. 10, p. 128, 2013.

S. Mukkamala, G. Janoski, and A. Sung, "Intrusion detection using neural networks and support vector machines," in *Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on*, vol. 2. IEEE, 2002, pp. 1702–1707.

D. Narsingyani and O. Kale, "Optimizing false positive in anomaly based intrusion detection using genetic algorithm," in *MOOCs, Innovation and Technology in Education (MITE), 2015 IEEE 3rd International Conference on*. IEEE, 2015, pp. 72–77.

W. S. Noble, "What is a support vector machine?" *Nature biotechnology*, vol. 24, no. 12, pp. 1565–1567, 2006.

H. Om and A. Kundu, "A hybrid system for reducing the false alarm rate of anomaly intrusion detection system," in *Recent Advances in Information Technology (RAIT), 2012 1st International Conference on*. IEEE, 2012, pp. 131–136.

S. Omar, A. Ngadi, and H. H.Jebur, "Machine learning techniques for anomaly detection: An overview," *International journal of computer applications*, vol. 79, no. 2, 2013.

E. Parzen, "On estimation of a probability density function and mode," *The annals of mathematical statistics*, vol. 33, no. 3, pp. 1065–1076, 1962.

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

Ponemon and Damballa, "The cost of malware containment," 2015.

Ponemon and IBM, "2016 cost of data breach study: Global analysis," 2016.

B. Schölkopf and A. J. Smola, *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural computation*, vol. 13, no. 7, pp. 1443–1471, 2001.

H. J. Shin, D.-H. Eom, and S.-S. Kim, "One-class support vector machines—an application in machine fault detection and classification," *Computers & Industrial Engineering*, vol. 48, no. 2, pp. 395–408, 2005.

D. M. J. Tax, "One-class classification," Ph.D. dissertation, TU Delft, Delft University of Technology, 2001.

D. Tax, "Ddtools, the data description toolbox for matlab," June 2015, version 2.1.2.

R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.

K. M. Ting, "Precision and recall," in *Encyclopedia of machine learning*. Springer, 2011, pp. 781–781.

V. Vapnik and A. Sterin, "On structural risk minimization or overall risk in a problem of pattern recognition," *Automation and Remote Control*, vol. 10, no. 3, pp. 1495–1503, 1977.

V. N. Vapnik, *Statistical learning theory*. Wiley New York, 1998, vol. 1.

F. Xiao and X. Li, "Using outlier detection to reduce false positives in intrusion detection," in *Network and Parallel Computing, 2008. NPC 2008. IFIP International Conference on.* IEEE, 2008, pp. 26–33.

S. Yin, X. Zhu, and C. Jing, "Fault detection based on a robust one class support vector machine," *Neurocomputing*, vol. 145, pp. 263–268, 2014.

M. Yousef, S. Jung, L. C. Showe, and M. K. Showe, "Learning from positive examples when the negative class is undetermined-microrna gene identification," *Algorithms for Molecular Biology*, vol. 3, no. 1, p. 2, 2008.

A. Ypma and R. P. Duin, "Support objects for domain approximation." ICANN, 1998.

M. Zhang, B. Xu, and J. Gong, "An anomaly detection model based on one-class svm to detect network intrusions," in *2015 11th International Conference on Mobile Ad-hoc and Sensor Networks (MSN).* IEEE, 2015, pp. 102–107.

ANNEXE A NUMERICAL EXAMPLE

In this annex, we discuss how exactly the outliers are calculated using one-class SVM. To do so, we use one 2D toy example of ten observations ($n = 10$) such that 1 of them is an outlier and the rest belongs to the target class. Table A.1 shows the values of these observations. Note that all the equations provided in this appendix have been already explained in chapter 3, however, for convenience, we provide them again.

Table A.1 Observations

	X1	X2	Result
1	2	1	1
2	20	20	-1
3	1	1	1
4	1	2	1
5	3	2	1
6	2	3	1
7	3	0	1
8	3	1	1
9	0	1	1
10	2	0	1

Here, the value of ν is considered equal to 0.2. We assume that the RBF kernel is used with $\gamma = 0.05$. As such, we can calculate the kernel matrix as depicted in table A.2. These values are computed based on the equation A.1. For example $K(\langle x_1, x_3 \rangle) = exp(-\gamma \sum_{j=1}^{p}(x_{1j} - x_{3j})^2) = exp(-0.05 * ((2-1)^2 + (1-1)^2)) = 0.951$. p is the number of dimensions that in here it is equal to 2.

$$K(\langle x_i, x_k \rangle) = exp(-\gamma \sum_{j=1}^{p}(x_{ij} - x_{kj})^2). \tag{A.1}$$

For outlier detection task based on one-class SVM algorithm, first, we need to train the model, i.e. compute the one-class classifier. Then, we can use the calculated frontier for detection of outliers in the training set. Recall that α_i is the dual coefficient of ith observation, and K_{ij} is the dot product between observation i and observation j.

Table A.2 Kernel matrix

	1	2	3	4	5	6	7	8	9	10
1	1		0.951	0.905	0.905	0.819	0.905	0.951	0.819	0.951
2	0	1	0	0	0	0	0	0	0	0
3	0.951	0.	1.	0.951	0.779	0.779	0.779	0.819	0.951	0.905
4	0.905	0.	0.951	1.	0.819	0.905	0.67	0.779	0.905	0.779
5	0.905	0.	0.779	0.819	1.	0.905	0.819	0.951	0.607	0.779
6	0.819	0.	0.779	0.905	0.905	1.	0.607	0.779	0.67	0.638
7	0.905	0.	0.779	0.67	0.819	0.607	1.	0.951	0.607	0.951
8	0.951	0.	0.819	0.779	0.951	0.779	0.951	1.	0.638	0.905
9	0.819	0.	0.951	0.905	0.607	0.67	0.607	0.638	1.	0.779
10	0.951	0.	0.905	0.779	0.779	0.638	0.951	0.905	0.779	1.

Hence, we perform the following steps to initialize the algorithm:

1. we randomly initialize 20 percent of training points to the value of $\frac{1}{\nu l} = \frac{1}{0.2*10} = 0.5$. Suppose that we choose $\alpha_2 = 0.5$ and $\alpha_7 = 0.5$. Hence, at iteration 1 the values of vector α is as follows:

   $[0, 0.5, 0, 0, 0, 0, 0.5, 0, 0, 0]$

2. we calculate the output vector based on the following equation:

$$O_i = K_{2i}\alpha_2 + K_{7i}\alpha_7 + C_i. \tag{A.2}$$

   Moreover, C_i is calculated by the equation $C_i = \sum_{t\neq 2,7}^{l} \alpha_t K_{i,t}$. In equation A.2, the values of vector C_i are zero, since all the dual coefficients except 2 and 7 are zero. Now, we can calculate the values of the output vector for all the observations, as follows:
   $O_1 = K_{21}\alpha_2 + K_{71}\alpha_7 + C_1 = 0 * 0.5 + 0.905 * 0.5 + 0 = 0.453$
   $O_2 = K_{22}\alpha_2 + K_{72}\alpha_7 + C_2 = 1 * 0.5 + 0 * 0.5 + 0 = 0.5$
   $O_3 = K_{23}\alpha_2 + K_{73}\alpha_7 + C_3 = 0 * 0.5 + 0.779 * 0.5 + 0 = 0.39$
   $O_4 = K_{24}\alpha_2 + K_{74}\alpha_7 + C_4 = 0 * 0.5 + 0.67 * 0.5 + 0 = 0.335$
   $O_5 = K_{25}\alpha_2 + K_{75}\alpha_7 + C_5 = 0 * 0.5 + 0.819 * 0.5 + 0 = 0.449$
   $O_6 = K_{26}\alpha_2 + K_{76}\alpha_7 + C_6 = 0 * 0.5 + 0.607 * 0.5 + 0 = 0.303$
   $O_7 = K_{27}\alpha_2 + K_{77}\alpha_7 + C_7 = 0 * 0.5 + 1 * 0.5 + 0 = 0.5$
   $O_8 = K_{28}\alpha_2 + K_{78}\alpha_7 + C_8 = 0 * 0.5 + 0.951 * 0.5 + 0 = 0.475$
   $O_9 = K_{29}\alpha_2 + K_{79}\alpha_7 + C_9 = 0 * 0.5 + 0.607 * 0.5 + 0 = 0.39$

$O_10 = K_{210}\alpha_2 + K_{710}\alpha_7 + C_{10} = 0 * 0.5 + 0.951 * 0.5 + 0 = 0.475$

3. we initialized $\rho = max\{O_i : i\epsilon[l], \alpha_i > 0\} = max(O_2, O_7) = 0.5$.

Once the initialization of the algorithm is done, we can optimize the dual coefficients based on the following steps:

1. we select the first pair of dual coefficients (α_i) based on KKT conditions:

$$(O_i - \rho).\alpha_i > 0 \tag{A.3}$$

$$(\rho - O_i).(\frac{1}{vl} - \alpha_i) > 0. \tag{A.4}$$

Based on the equations A.3 and A.4, we can find the observations that violate these conditions, as follows:

$(O_1 - \rho).\alpha_1 > 0 = (0.453 - 0.5).0 > 0 = 0 > 0 = False$
$(\rho - O_1).(\frac{1}{\nu l} - \alpha_1) > 0 = (0.5 - 0.453).(0.5 - 0) > 0 = 0.113 > 0 = True$

$(O_2 - \rho).\alpha_1 > 0 = (0.5 - 0.5).0 > 0 = 0 > 0 = False$
$(\rho - O_2).(\frac{1}{\nu l} - \alpha_2) > 0 = (0.5 - 0.5).(0.5 - 0.5) > 0 = 0.113 > 0 = False$

The rest of these calculation is presented in table A.3 at iteration 1. Among all the observations that violate the KKT conditions, we choose the first observation.

2. we select second dual coefficient (α_j) for optimization based on the following equation:

$$j = argmax_{n\epsilon SV_{nb}} \|O_i - O_j\|. \tag{A.5}$$

In equation A.5, SV_{nb} has the indexes of SV observations. In the first iteration, this set is empty, thus, we choose the second dual coefficient based on its KKT output. Consider that we choose observation 7. If this set is not empty, we need to choose the one that has the maximum difference of its output to the members of SV_{nb} set based on equation A.5.

3. we calculate $\Delta = \alpha_i + \alpha_j = \alpha_1 + \alpha_7 = 0 + 0.5 = 0.5$.

4. we update the value of α based on following equation:

$$\alpha_{jtemp} = \alpha_j^* + \frac{O_i - O_j}{K_{jj} + K_{ii} - 2K_{ji}} \tag{A.6}$$

In equation A.6, α_j^* is the previous α_j value.

So, the new value of α_{7temp} is calculated as follows:

$\alpha_{7temp} = \alpha_7^* + \frac{O_1 - O_7}{K_{11}+K_{77}-2K_{17}}.0.5 + \frac{0.453-0.5}{1+1-2*0.905} = 0.5 + \frac{0.047}{0.19} = 0.253$

Before updating the value of α_7, we need to calculate $L = max(0, (\Delta - \frac{1}{vl})) = max(0, 0) = 0$ and $H = min(\frac{1}{vl}, \Delta) = min(0.5, 0.5) = 0.5$

Now, we need to select the value of α_7 based on the following equation:

$\alpha_7 = min(max(L, \alpha_{7temp}), H) = min(max(0, 0.253), 0.5) = 0.253$

5. we modify α_1's value ($\alpha_i = \Delta - \alpha_j = \Delta - \alpha_7 = 0.5 - 0.253 = 0.247$).

6. we update parameter $\rho = max\{O_i : i\epsilon[l], \alpha_i > 0\} = 0$

7. we add all the observations with $0 < \alpha_t < \frac{1}{\nu l}, t\epsilon\{1, .., 10\}$ to SV_{nb} set.

   Hence, observations 1 and 7 are added to SV_{nb} set, since their dual coefficient values are between 0 and 0.5.

8. we update vector O based on equation:

$$O[x_i] = \left(\sum_i \alpha_i k(\mathbf{x}_i, \mathbf{x}) - \rho \right). \tag{A.7}$$

These steps should iterate until there is no observation that violates any KKT conditions.

Once the algorithm has calculated the dual coefficients and the value of intercept (ρ), we can calculate the output of decision function for all the training points based on equation A.8. This equation is the equation of the frontier.

$$f(\mathbf{x}) = sgn\left(\sum_i \alpha_i k(\mathbf{x}_i, \mathbf{x}) - \rho \right). \tag{A.8}$$

.

In table A.3 , we summarized the optimization algorithm. The results of 13 iterations have shown, however, we illustrated only the important variables such as Lagrangian parameter α, the output of each iteration and the calculated outputs of KKT conditions (C_1, C_2).

0.7

Table A.3 – Optimization table

V	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
α_{iter1}	0	0.5	0	0	0	0	0.5	0	0	0
O_{iter1}	0.453	0.5	0.39	0.335	0.409	0.303	0.5	0.475	0.303	0.475
C1-C2	F-T	F-F	F-T	F-T	F-T	F-T	F-F	F-T	F-T	F-T
α_{iter2}	0.2471	0.5	0	0	0	0	0.253	0	0	0
Output	-0.024	0	-0.068	-0.107	-0.069	-0.144	-0.023	-0.025	-0.144	-0.025
C1-C2	F-T	F-F	F-T	F-T	F-T	F-T	F-F	F-T	F-T	F-T
α_{iter3}	0	0.5	0	0	0	0.247	0.253	0	0	0
O_{iter2}	-0.069	0	-0.111	-0.107	-0.07	-0.1	-0.097	-0.067	-0.181	-0.102
C1-C2	F-T	F-F	F-T	F-T	F-T	F-T	F-T	F-T	F-T	F-T
α_{iter4}	0	0.452	0	0	0	0.247	0.301	0	0	0
Output	0.475	0.452	0.427	0.425	0.47	0.43	0.451	0.479	0.348	0.444
C1-C2	F-F	F-F	F-T	F-T	F-F	F-T	F-T	F-F	F-T	F-T
α_{iter5}	0	0.452	0	0	0	0.123	0.301	0	0.124	0
O_{iter5}	0.023	0	-0.004	-0.027	-0.019	-0.063	-0.001	0.009	-0.063	0.009
C1-C2	F F	F F	F-T	F-T	F-T	F-T	F-T	F-F	F-T	F-F
α_{iter6}	0	0.452	0	0	0	0.123	0.301	0	0.124	0
Output	0.475	0.452	0.448	0.425	0.433	0.389	0.451	0.461	0.389	0.461
C1-C2	F-F	F-F	F-T	F-T	F-T	F-T	F-T	F-F	F-T	F-F
α_{iter7}	0	0.452	0	0	0	0.202	0.222	0	0.124	0
Output	0.016	0	-0.004	-0.008	-0.012	-0.032	-0.032	-0.004	-0.058	-0.015
C1-C2	F-F	F-F	F-T	F-T	F-T	F-T	F-T	F-T	F-T	F-T
α_{iter8}	0	0.452	0	0	0	0.202	0.222	0	0.124	0
Output	0.468	0.452	0.448	0.444	0.44	0.42	0.42	0.448	0.394	0.437
C1-C2	F-F	F-F	F-T	F-T	F-T	F-T	F-T	F-T	F-T	F-T
α_{iter9}	0	0.423	0	0	0	0.202	0.222	0	0.153	0
Output	0.04	-0.029	0.024	0.018	0.005	-0.013	-0.015	0.014	-0.029	0.007
C1-C2	F-F	F-T	F-F	F-F	F-F	F-T	F-T	F-F	F-T	F-F
α_{iter10}	0	0.423	0	0	0	0.202	0.222	0	0.153	0
Output	0.492	0.423	0.476	0.47	0.457	0.439	0.437	0.466	0.423	0.459
C1-C2	F-F	F-T	F-F	F-F	F-F	F-F	F-T	F-F	F-T	F-F
α_{iter11}	0	0.423	0	0	0	0.202	0.222	0	0.153	0
Output	0.053	-0.016	0.037	0.031	0.018	0	-0.002	0.027	-0.016	0.02

Continued on next page

Table A.3 – Optimization table (continued and end)

V	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
C1-C2	F-F	F-T	F-F	F-F	F-F	F-F	F-T	F-F	F-T	F-F
α_{iter12}	0	0.423	0	0	0	0.202	0.222	0	0.153	0
Output	0.492	0.423	0.476	0.47	0.457	0.439	0.437	0.466	0.423	0.459
C1-C2	F-F	F-T	F-F	F-F	F-F	F-F	F-T	F-F	F-T	F-F
α_{iter13}	0	0.423	0	0	0	0.202	0.222	0	0.153	0
Output	0.053	-0.016	0.037	0.031	0.018	0	-0.002	0.027	-0.016	0.02
C1-C2	F-F	F-T	F-F	F-F	F-F	F-F	F-T	F-F	F-T	F-F

Based on table A.3, observations $2, 6, 7, 9$ are selected as SVs, i.e. observations that have impact on determination of the output for each observations with dual coefficient greater than zero.

The last row of this table depicts the output of decision function at iteration 13 for all of the ten training observations. Now, we can classify all the training points based on the following equation:

$$h(x) = \begin{cases} Target \quad if \quad f(x) \geq 0 \\ Outlier \quad if \quad f(x) < 0 \end{cases} \tag{A.9}$$

Based on equation A.9, at iteration 13, observations $2, 7, 9$ are outliers and the rest are target observations. Obviously, observations 6 and 8 are false alarms.

It should be noted that there are more iterations needed for this algorithm to have no observations that violates the KKT conditions. Note that the main purpose of this example is to show the exact steps of this algorithm based on the information provided in Schölkopf et al. (2001). If we use LIBSVM, we end up with different dual coefficient values and different intercept value but the same set of $SV_{nb} = 2, 6, 7, 9$.

ANNEXE B EMPIRICAL RESULT OF ROBUST SCALING

In this annex, we present the result of scaling data with robust scalar instead of feature scaling to $[0, 1]$. In this case, we scale all the training points without removing attack patterns. Here, we used the same method of feature selection visually (3 features). Table B.1 shows the selected values for hyper-parameters of one-class SVM algorithm. Based on the result, we select $\nu = 0.004$ and $\gamma = 0.05$ for training the model.

Table B.1 Selected parameters of one-class SVM in 10 different runs (robust scaling)

	avgF-measure	avgFN	avgFP	avgAUC	ν	γ
Run # 1	0.106	8	76	0.686	0.002	0.25
Run # 2	0.101	9	62	0.648	0.001	0.05
Run # 3	0.114	9	53	0.649	0.001	0.15
Run # 4	0.115	8	70	0.686	0.004	0.05
Run # 5	0.116	7	72	0.696	0.004	0.05
Run # 6	0.124	7	66	0.696	0.004	0.15
Run # 7	0.097	8	76	0.676	0.002	0.05
Run # 8	0.098	8	76	0.667	0.001	0.05
Run # 9	0.128	7	72	0.715	0.004	0.05
Run # 10	0.119	7	72	0.706	0.005	0.05

According to table B.1, we can conclude that feature scaling to $[0, 1]$ is a better choice comparing robust scalar (cf 4.1). F-measure values are much higher comparing feature scaling method.

ANNEXE C EMPIRICAL RESULT OF LASSO REGRESSION AS FEATURE SELECTION METHOD

In this annex, we explore the result of using Lasso regression as feature selection method. With this technique, we end up choosing two features (CPU Voltage and CPU Usage) as the selected features. Table C.1 depicts the result of 10 different runs on the real dataset for model selection. Based on this empirical results, we can conclude that selecting features visually has a better performance comparing Lasso regression.

Table C.1 Selected parameters of one-class SVM in 10 different runs (Lasso)

	avgF-measure	avgFN	avgFP	avgAUC	ν	γ
Run # 1	0.186	11	4	0.567	0.001	0.25
Run # 2	0.158	11	4	0.567	0.001	0.05
Run # 3	0.133	11	5	0.548	0.001	0.15
Run # 4	0.117	11	5	0.557	0.001	0.05
Run # 5	0.155	11	4	0.557	0.001	0.05
Run # 6	0.157	11	4	0.567	0.001	0.15
Run # 7	0.17	11	5	0.558	0.001	0.05
Run # 8	0.153	11	4	0.558	0.001	0.05
Run # 9	0.124	11	5	0.548	0.001	0.05
Run # 10	0.172	11	4	0.567	0.001	0.05