SEMI-SUPERVISED AND SELF-EVOLVING LEARNING ALGORITHMS WITH APPLICATION TO ANOMALY DETECTION IN CLOUD COMPUTING

Husanbir Singh Pannu, M.S., B.Tech.

Dissertation Prepared for the Degree of

DOCTOR OF PHILOSOPHY

UNIVERSITY OF NORTH TEXAS

December 2012

APPROVED:

Jianguo Liu, Major Professor
Song Fu, Committee Member
Joseph Iaia, Committee Member
Su Gao, Chair of the Department of Mathematics
Mark Wardell, Dean of the Toulouse Graduate School
Semi-supervised learning (SSL) is the most practical approach for classification among machine learning algorithms. It is similar to the humans way of learning and thus has great applications in text/image classification, bioinformatics, artificial intelligence, robotics etc. Labeled data is hard to obtain in real life experiments and may need human experts with experimental equipments to mark the labels, which can be slow and expensive. But unlabeled data is easily available in terms of web pages, data logs, images, audio, video les and DNA/RNA sequences. SSL uses large unlabeled and few labeled data to build better classifying functions which acquires higher accuracy and needs lesser human efforts. Thus it is of great empirical and theoretical interest. We contribute two SSL algorithms (i) adaptive anomaly detection (AAD) (ii) hybrid anomaly detection (HAD), which are self evolving and very efficient to detect anomalies in a large scale and complex data distributions. Our algorithms are capable of modifying an existing classifier by both retiring old data and adding new data. This characteristic enables the proposed algorithms to handle massive and streaming datasets where other existing algorithms fail and run out of memory. As an application to semi-supervised anomaly detection and for experimental illustration, we have implemented a prototype of the AAD and HAD systems and conducted experiments in an on-campus cloud computing environment. Experimental results show that the detection accuracy of both algorithms improves as they evolves and can achieve 92.1% detection sensitivity and 83.8% detection specificity, which makes it well suitable for anomaly detection in large
and streaming datasets. We compared our algorithms with two popular SSL methods (i) subspace regularization (ii) ensemble of Bayesian sub-models and decision tree classifiers. Our contributed algorithms are easy to implement, significantly better in terms of space, time complexity and accuracy than these two methods for semi-supervised anomaly detection mechanism.
Copyright 2012

by

Husanbir Singh Pannu
First and foremost, I would like to thank Dr. Jianguo Liu, my major professor for all his hard work in guiding me through the completion of my research project. I would also like to thank Dr. Song Fu for all his patience, time and scholarly insight provided throughout this research. This work is the result of their guidance, support and constant encouragement. They have set a high academic and professional standard for me to follow. I am also grateful to Dr. Iaia for assisting me in the preparation for the qualifying exams and overcoming the stressful times.

Appreciation is extended to IEEE International Symposium on Reliable Distributed Systems (SRDS 2012), IEEE Global Communications (GlobeCom 2012), Advanced Data Mining and Applications (ADMA 2012) Springer publication, IEEE International Performance Computing and Communications Conference (IPCCC 2012). Contents regarding AAD and HAD algorithms in sections 3.3, 3.4 and chapters 5 and 6 have been accepted for publication by these four conferences. I am using them in my dissertation with their permission.

I have great appreciation for all the committee members for taking the time to read the manuscript and providing valuable feedback.

I am grateful to my father, mother and wife Kulwinder for always being there for me during this long journey and never leaving my side.

Last but not least, I would like to thank all the faculty and staff members of Department of Mathematics for giving me the opportunity to study here and broaden my horizons. I would not be here today without the support of all these individuals.
# CONTENTS

ACKNOWLEDGMENTS iii

LIST OF TABLES vii

LIST OF FIGURES viii

CHAPTER 1. MACHINE LEARNING AND SEMI-SUPERVISED LEARNING 1

1.1. Semi-Supervised Learning Motivation 2

1.2. Self Evolving Algorithms 3

1.3. Introduction to Anomaly Detection 4

1.4. Related Work 5

CHAPTER 2. ONE AND TWO CLASS SUPPORT VECTOR MACHINES 8

2.1. Two-Class SVM 8

2.1.1. Linear Two-Class SVM 8

2.1.2. Penalized Linear Two-Class SVM 11

2.1.3. Non Linear Two-Class SVM 12

2.2. One-Class SVM 14

2.2.1. Separating Data From Origin and Minimum Enclosing Sphere 15

2.3. Two-Class SVM and One-Class SVM Comparison 18

CHAPTER 3. SEMI-SUPERVISED LEARNING ALGORITHMS 19

3.1. Active and Semi-supervised Data Domain Description Algorithm 19

3.2. Semi-supervised Classification by Homogeneous Cluster Labeling 23

3.2.1. Non Overlapping Substantive Clusters (Case 1) 26

3.2.2. Automatic Method to Find Best RBF Kernel Parameter for SVM 27

3.2.3. Non Overlapping Grouped Clusters (Case 2) 29

3.2.4. Overlapping Clusters (Case 3) 29
3.3. Semi-Supervised Adaptive Anomaly Detection (AAD) 32
3.3.1. The Objective Function 33
3.3.2. AAD Algorithm 34
3.4. Hybrid Anomaly Detection (HAD) 36
3.4.1. Detector Determination 38
3.4.2. Detector Retraining and Working Data Set Selection 39
3.4.3. Sample Margin Information For Updating Working Dataset 40
3.4.4. HAD Algorithm 42

CHAPTER 4. INTRODUCTION TO CLOUD COMPUTING 45
4.1. Definition 46
4.2. Essential Characteristics 46
4.3. Service Models 47
4.4. Deployment Models 48

CHAPTER 5. SYSTEM OVERVIEW AND CLOUD METRIC EXTRACTION 50
5.1. System Overview 50
5.2. System Design 51
5.2.1. Cloud Metric Extraction 52

CHAPTER 6. APPLICATIONS OF AAD AND HAD TO ANOMALY
DETECTION IN CLOUD COMPUTING 54
6.1. Experiment Settings 54
6.2. Cloud Metric Extraction Results 55
6.2.1. AAD Performance Evaluation 59
6.2.2. Conclusion 62
6.2.3. HAD Performance Evaluation 62
6.2.4. Conclusion 64

CHAPTER 7. CONCLUSION 65
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Few popular kernel functions</td>
<td>14</td>
</tr>
<tr>
<td>3.1</td>
<td>Definition of USV, BSV and NSV</td>
<td>24</td>
</tr>
<tr>
<td>3.2</td>
<td>Comparision of time complexities between BB and HCL</td>
<td>26</td>
</tr>
<tr>
<td>6.1</td>
<td>CPU and SWAP statistics, I/O requests</td>
<td>56</td>
</tr>
<tr>
<td>6.2</td>
<td>I/O requests (continued), memory page statistics, %memused</td>
<td>57</td>
</tr>
<tr>
<td>6.3</td>
<td>Network statistics</td>
<td>58</td>
</tr>
<tr>
<td>6.4</td>
<td>14 most prominent metrics</td>
<td>59</td>
</tr>
</tbody>
</table>
### LIST OF FIGURES

1.1 Supervised learning versus semi-supervised learning. $O$ and $+$ are labeled data and dots are unlabeled data

2.1 Either plane could separate the data but not with maximum margin

2.2 Dotted line shows the maximum margin SVM hyperplane [11]

2.3 The separating hyperplane [32]

2.4 Mapping the data by $\phi(x_1, x_2) \rightarrow (x_1^2, \sqrt{2}x_1x_2, x_2)$

2.5 Separating hyperplane and minimum enclosing sphere are equivalent [57] and called One-Class SVM

2.6 Mapping the data into feature space to find minimum enclosing sphere

3.1 Differential approximation of Hinge loss is Huber loss. $\Delta = 1$ and $\epsilon = 0.5$

3.2 SVC data description phase: Mapping the data into feature space, find minimum enclosing sphere, map the sphere back into original space, it splits into contours enclosing the clusters [27]

3.3 SVDD data description

3.4 Label support vectors

3.5 Ignore unlabeled data

3.6 Draw SVM plane for the labeled data

3.7 Graph of $\sigma$ vs $J(\sigma)$ for two moon case Fig3.5

3.8 Two principal components of IRIS Dataset

3.9 Three cluster cores of IRIS

3.10 Cluster frequency histogram obtained from SVC
3.11 Minimum enclosing sphere illustration in the Gaussian kernel feature space
(circles represent anomalies and crosses are normal data points) 33

3.12 The candidates of removable NSV and ∈ region 41

4.1 Cloud computing infrastructure [10] 45
4.2 Visual model of cloud computing by National Institute of Standards and Technology [2] 46

5.1 A dependable cloud computing infrastructure. 50

6.1 Redundancy and relevance among cloud metrics. 55
6.2 Results from cloud metric extraction. 55
6.3 Contour for anomaly detection after AAD-adaptation I. 60
6.4 Contour for anomaly detection after AAD-adaptation III. 60
6.5 Anomaly detection results with self-evolvement III. Only 10% of the detected normal data points are plotted due to the size of the figure and the readability. 61
6.6 Sensitivity and specificity of anomaly detection improves with AAD’s self adaptations. 61
6.7 Contours of decision function during Adaption-I 63
6.8 Contours of decision function during Adaption-II 63
6.9 Credibility scores improvement and stabilization of 1-class and 2-class SVM during hybridization 63
6.10 Sensitivity and specificity of hybrid anomaly detection algorithm (HAD) with adaptation 63
CHAPTER 1

MACHINE LEARNING AND SEMI-SUPERVISED LEARNING

The term “machine learning” was introduced by Arthur Samuel in 1959. It is the process that searches through the data to look for the patterns, extract information and use that to adjust/improve the program’s own actions and understanding accordingly. It is a self evolving paradigm. For example if you search a product in Google, it will pull up the product closest to the individual’s location. It will also recognize any past searches to pull up the information that might be of interest to the client. Machine learning has three major categories of algorithms, supervised, unsupervised and semi-supervised. Supervised learning calculates a function $F$ from the labeled data and then performs testing for future unlabeled data, e.g. separating apples from bananas. Unsupervised learning searches for patters in data distribution to learn $F$ when the data is unlabeled, e.g. dividing the population according to their salary. Semi-supervised learning uses both labeled and unlabeled data to find $F$ i.e. learning from the class labels and distribution of the data to get better generalization. We focus on semi-supervised learning (SSL) algorithms.

There are many paradigms which fit in SSL and most familiar one is called semi-supervised classification where we do learning particularly for classification for $L$ labeled and $U$ unlabeled data points with $U >> L$ [75]. For example we have to train somehow

![Figure 1.1. Supervised learning versus semi-supervised learning. O and + are labeled data and dots are unlabeled data](image)
a classifier $\mathcal{F}$ which would be better than the classifier found using only the labeled data (figure 1.1). Second category is semi-supervised clustering, where we start from clustering and then do constrained clustering by putting some domain knowledge. We start with unlabeled data and use supervised information in terms of labels or must-link and cannot-link constraints. The goal here is to do better clustering. Third category is semi-supervised regression which models the relationship between two or more random variables using both labeled and unlabeled data and used for predicting the value of dependent variable for given independent variables. Our research focus is on the first category, i.e. semi-supervised classification.

1.1. Semi-Supervised Learning Motivation

Two basic motivations for semi-supervised learning (SSL) are:

(1) Labeled data is hard to get and may need human experts/annotators to mark the labels which can be slow and expensive. In some cases we may have to do experiments needing the devices which could be expensive. So labeled data is expensive and difficult to collect, but unlabeled data is easily available so why not to use it?

(2) From cognitive science prospective, the human learning in general is similar to SSL. Imagine the concept learning in children when they see an object $x$ and want to get concept $y$. Sometimes they receive supervised information when we point to the animal and say Dog, which works as an explicit labeled data instance. But they also see other animals walking around by themselves and they might be using that information somehow too. Thus SSL models would be used as computation model for human learning.

As an example of how labeling is hard, consider speech recognition where we want to label the speech string (acoustic signal) at a fine grain level. We need human annotators to sit and listen to the recordings. The word “film” has following annotations: f ih.n uh.gl.n m (the phonetic units) and “be all” = bcl b iy iy_tr ao_tr ao l dl. It takes 400 hours to transcribe 1 hour of speech. Similarly while doing parsing of sentence, we need training data e.g. parse
trees which are initially created by linguistics by hand and quite slow to create. Unlabeled
data is cheap to collect so why not to enhance the performance with little extra price.

In SSL there are two types of unlabeled data, one is used during training and another is
used for testing. SSL is inductive learning in contrast to transductive learning [21]. Induction
means that if we are given labeled and unlabeled data set, then we want to create a classifier
\( \mathcal{F} \) which should be a good predictor on a future unlabeled testing data. But in transductive
learning, no worries about the testing unlabeled data in future. The secret of learning from
unlabeled data in SSL are the *assumptions* that we have to make, e.g. cluster assumption
and manifold assumption. In cluster assumption we assume that points in the same cluster
(a dense region) should have similar labels. Manifold assumption says that there exists
a lower dimensional manifold where almost all the data lives, for example the black and
white photos are made by just the intensity variation of two colors. We may also need
to make an assumption about the data distribution, e.g. semi-supervised learned boundary
using two labeled points and assuming that unlabeled data came from Gaussian distribution.
The labeled data may not belong to center the distribution. We use Gaussian distribution
assumption for image classification, Naive Bayes for text categorization and Hidden Markov
Models for speech recognition problems.

1.2. Self Evolving Algorithms

The idea of self-evolving learning is natural to humans learning process i.e. learning
from mistakes. World’s first self evolving program was Samuel Checkers playing which was
invented in 1955. It illustrated the basic idea of artificial intelligence. There are a lot
of self evolving algorithms used in artificial intelligence [42], robotics [15], neural networks
[71], fuzzy control systems [51] and optimizations [52]. For example in artificial intelligence,
the decision-tree based learning methods use tree like structures, organize the elements and
simplify the information flow among the entities. Neural networks is also self evolving system
which modifies itself depending upon internal and external information traveling into the
network. However, we are developing a new framework to use self evolving mechanism in
machine learning to detect the anomalies/outliers.
1.3. Introduction to Anomaly Detection

The process of detecting the abnormality in otherwise normal, expected or standard data distribution is called anomaly or outliers detection. Anomalies arise due to various reasons such as mechanical faults, changes in system behavior, fraudulent behavior, anomalous gene clusters or pathogenic islands in bacterial genomes (in bioinformatics), human/instrument error in experiments. It has large application areas including bioinformatic, medical science for diagnosis, financial transactions, laboratory experiments, cloud computing to detect resource failure etc. Detection of anomalies can help administrators to take preventive measures before they escalate and also may enable detection of new attacks. With ever-growing complexity and dynamics of real time monitoring systems, anomaly detection is an effective approach to enhance system dependability [55]. It forecasts future failure occurrences in the system using runtime execution states of the system and the history information of observed failures. It provides valuable information for resource allocation, computation reconfiguration and system maintenance [43].

Most of the existing anomaly detection methods are based on statistical learning techniques [56]. They use supervised learning models to approximate the dependency of failure occurrences on various performance features [5], [55]. The underlying assumption of those methods is that the training dataset is labeled, i.e. for each data point used to train a predictor, the designer knows if it corresponds to a normal execution state or an anomaly. However, the labeled data is not always available in real-world situations, especially for newly deployed or managed systems. How to accurately forecast failure occurrences in such systems is challenging. There are also unsupervised learning methods that use no failure information such as support vector clustering [4], [8].

Numerous anomaly detection algorithms have been proposed and investigated. Chandola et al. [8] categorized them into six techniques: classification based, clustering based, statistical, nearest neighbor based, information theoretic, and spectral. Those algorithms can also be put into three groups: supervised (used when all the training data records are

Section 1.3 is also presented in my accepted papers [45], [46], [41], and [40] with Springer publication.
labeled and they include both normal and abnormal records), semi-supervised (used when some of the training data records are labeled, or when all the labeled records are normal), and unsupervised (used when the training data records are unlabeled).

Nearest neighbor based, information theoretic, and spectral methods do not have a training phase. But they can be slow for testing. Classification based, clustering based, and statistical methods are computationally expensive for training. But they are very fast for testing. In addition, training can be performed off line. Anomaly detection requires fast and accurate on line calculation. Therefore, classification based and clustering based methods are the choices for our anomaly detector. For now, statistical methods are not included since their performance heavily relies on certain data distribution assumptions.

We are developing a new framework which is semi-supervised and self-evolving learning algorithms with application to anomaly detection in large and streaming data sets. There are several approaches for classification and clustering. Our preliminary experiments show that support vector machines (SVM) and one-class support vector machines (support vector clustering (SVC) or support vector data description (SVDD)) work quite well. Reference for SVM and oneclass SVM are in Vapnik [68], Scholkopf et al. [59], [4] and Tax and Duin [65]. We focus on these two methods and show how to use them for self evolving semi-supervised anomaly detection. An ensemble incorporated with other methods, such as k-means, neural networks, Bayesian networks, and rule based, can easily be designed using the same framework.

1.4. Related Work

The online detection of anomalous system behavior [8] caused by operator errors [44], hardware/software failures [50], resource over-/under-provisioning [35],[34] and similar causes is a vital element of operations in large-scale data centers and safety critical systems. For instance the ever-increasing computer network operation scale coupled with the increasing complexity of software, applications, and workload patterns, anomaly detection methods

Section 1.4 is also presented in my accepted papers [45], [46], [41], and [40] with Springer publication.
must operate automatically at runtime and without the need for prior knowledge about normal or anomalous behaviors.

Anomaly detection based on analysis of performance logs has been the topic of numerous studies. Hodge and Austin [29] provided an extensive survey of anomaly detection techniques developed in machine learning and statistical domains. A structured and broad overview of extensive research on anomaly detection techniques was presented in [8].

Most often the task of anomaly detection can be solved by estimating a probability density of the normal data. For example, in [6] the density is estimated by a Parzen density estimator, whereas in [47] a Gaussian distribution is used. In [53] not only the normal data density is estimated, but also the failure density. The drawback of these approaches is that they are not resistant to the training data that only define the area of the target data but do not represent the complete density distribution. Vapnik argued that the estimation of the complete density instead of computing the boundary around a data set might require too much data and could result in bad descriptions [69]. In [67], anomaly detection was modeled as a classification problem, which was solved by convex programming. When the classification problem is not linearly separable, an ellipsoidal separation can be applied, where one of the classes is enclosed by an ellipsoid of minimum size. We use a similar method for anomaly detection.

Model-based approaches derive a probabilistic or analytical model of a system. A warning is triggered when a deviation from the model is detected [28]. Examples include an adaptive statistical data fitting method called MSET presented in [66], naive Bayesian based models for disk anomaly detection [26], and Semi-Markov reward models described in [22]. In large-scale systems, errors may propagate from one component to others, thereby making it difficult to identify the causes of failures. A common solution is to develop fault propagation models, such as causality graphs or dependency graphs [63]. Generating dependency graphs, however, requires a priori knowledge of the system structure and the dependencies among different components, which is hard to obtain in large-scale outliers detection.

Recently, data mining and statistical learning theories have received growing atten-
tion for anomaly detection and anomaly management. These methods extract anomaly patterns from systems’ normal behaviors, and detect abnormal observations based on the learned knowledge [49]. In [55], [70] the authors presented several methods to forecast anomalous events in IBM clusters. In [39],[38], Liang et al. examined several statistical methods for anomaly detection in IBM Blue Gene/L systems. In [36], Lan et al. investigated meta-learning based method for improving anomaly detection. In [20],[19], Fu and Xu developed a proactive failure management framework for networked computing systems. In [57], Scholkopf et al. applied an hyperplane to separate the target objects from the origin with maximal margin. This formulation is comparable with the support vector classifier in [67] and it is possible to define implicit mappings to obtain more flexible descriptions, which is explored for adaptive hybrid anomaly detection in this work.
Support vector machines (SVM) are a collection of supervised learning techniques that analyze data and recognize patterns. They are used for regression or classification of objects into two or more classes. SVM has two phases, training and testing. In case of simple binary classification problem, the training phase requires the \((data + binary\ labels)\) and calculates a separating boundary called the hyperplane, which divides the data into two classes. In the testing phase, SVM input the \(data\) and output their \(binary\ labels\) by comparing their locations with the separating hyperplane. The theory was developed by Vapnik and Chervonenkis and became highly popular in 90s because it is theoretically robust, computationally efficient, widely applicable in real world applications and easy to implement. SVM is based on statistical learning theory [68] and can be applied to a wide range of classification and regression applications, for example handwritten recognition, face detection in images, gene expression data classification in bioinformatics. SVM classification or regression problem becomes interesting when the data is high dimensional, large in size or it is not linearly separable. In our algorithm we have used two-class and one-class SVM which are discussed as follows.

2.1. Two-Class SVM

Two-Class SVM learns to separate the given data set by fitting a maximum margin hyperplane that divides both classes. It works well for linearly or non-linearly separable data but the data should be somewhat balanced. This means that the minor class should be at least 10% in population compared to the major class. Let’s discuss the linear, penalized linear and non-linear two-class SVM.

2.1.1. Linear Two-Class SVM

When the data is linearly separable, we need a classifier with smallest empirical risk, i.e. completely separating both classes. Secondly we need to choose a particular classifier
Figure 2.1. Either plane could separate the data but not with maximum margin.

Figure 2.2. Dotted line shows the maximum margin SVM hyperplane [11] from the family, that has maximum margin. In figure 2.1 the data is linearly separable and a line can separate the data $X \in \mathbb{R}^2$. Similarly a linear hyperplane can separate $X \in \mathbb{R}^d$ for $d > 2$. In figure 2.2 the two solid lines indicate maximum margin and the dotted line is the best decision boundary called SVM plane.

To fully grasp the mathematical concept, we require the knowledge of numerical optimization using Lagrange multipliers [23] and basic matrix theory. Let $T$ be the training set and $T = (X, Y)$ where $X = \{(x_i | x_i \in \mathbb{R}^d \}_{i=1}^m$ and $Y = \{y_i | y_i \in \{-1, 1\}\}_{i=1}^m$; $y_i$ is a binary variable and indicates the class of $x_i$ which is a $d$ dimensional vector. Our goal is to separate all points $x_i$ with $y_i = 1$ from $x_j$ with $y_j = -1 \ \forall \ i, j$ with maximum margin. For future testing, the SVM plane results in good generalization and least amount of error in classification. We can find SVM by solving a quadratic programming problem. The general form of SVM hyperplane is $f(x) = w.x + b = 0$ with following two properties:

- $w$ is normal to the hyperplane.
- $\frac{b}{||w||}$ is the perpendicular distance from the hyperplane to origin.

Support vectors (SVs) are the data points which are closest to the separating hyperplane and SVM aims to be as far as possible from SVs of both classes. They are indicated in the square boxes in figure 2.2. Finding/training SVM plane is in fact calculating $w$ and $b$ so that:
Figure 2.3. The separating hyperplane [32]

\[ f(x_i) \geq +1 \text{ for } y_i = +1 \quad \text{and} \quad f(x_i) \leq -1 \text{ for } y_i = -1 \]

In general we can say \( y_i.f(x) \geq 1 \forall i \). Each such hyperplane \((w, b)\) is a classifier that correctly separates all patterns from the training set. SVM training is followed by the testing phase, which inputs the data points and outputs the corresponding binary class:

\[
SVM_{\text{testing}}(x_i) = \begin{cases} 
+1 : f(x_i) > 0 \\
-1 : f(x_i) < 0 
\end{cases}
\]

We move forward to setup the SVM objective function by discussing following lemma:

**Lemma 3.1:** Prove that the distance between \( H_1 \) and \( H_2 \) in figure 2.3 is \( \frac{2}{\|w\|} \).

**Proof[32]:** For a linear classifier hyperplane \((H: w.x + b = 0)\) the distance between the origin and \( H \) is \( \frac{b}{\|w\|} \). Consider the patterns from class \(-1\) that satisfy the equality \( w.x + b = -1 \) and that determine the hyperplane \( H_1 \). Distance\((H_1,\text{origin}) = \frac{|-1-b|}{\|w\|} \). Similarly patterns from class \(+1\) satisfy the inequality \( w.x + b = +1 \) that determine hyperplane \( H_2 \). Thus Distance\((H_2,\text{origin}) = \frac{|1-b|}{\|w\|} \). Now \( H_1, H_2 \) and \( H \) are parallel and there is no data point between \( H_1 \) and \( H_2 \). Thus Distance\((H_1, H_2) = \frac{2}{\|w\|} \) Q.E.D.

The quadratic problem is to maximize \( \frac{2}{\|w\|} \) or minimize \( \frac{\|w\|^2}{2} \). Thus the primal objective function becomes:

\[
(1) \quad \text{minimize } \frac{\|w\|^2}{2} \text{ such that } y_i(w.x_i + b) \geq 1 \forall i
\]
2.1.2. Penalized Linear Two-Class SVM

In real world applications, even the linearly separable data may not be ideally linearly separable because of measurement errors or noise. When separating hyper-surface is almost linear, but with few exceptions, we can extend the above linear SVM methodology to handle data in such cases. We relax the constraints for Equation (1) slightly to allow for misclassified points by introducing slack variables \( \{\xi_i \geq 0\}_{i=1}^m \). In this soft margin SVM, data points on the incorrect side of the margin boundary have a penalty that increases with the distance from it. We are trying to reduce the number of misclassification’s by solving:

\[
\text{(2) } \quad \text{minimize } \frac{||w||^2}{2} + C \sum_{i=1}^m \xi_i \quad \text{such that } \quad y_i(w.x_i + b) + \xi_i \geq 1 \quad \text{and } \xi_i \geq 0 \quad \forall i
\]

The first goal is to maximize the margin and second is to minimize the number of slack variables \( \xi_i > 0 \). User defined parameter \( C \) controls the trade-off between the slack variable penalty and size of the margin. Decreasing the value of \( C \) means allowing more misclassifications and relaxing the hyperplane tension. To solve this constrained quadratic optimization problem we find its dual using Lagrange multipliers \( \alpha_i \geq 0 \) and \( \gamma_i \geq 0 \quad \forall i \) for the constraints. The primal Lagrangian function is

\[
\text{(3) } \quad L_P = \frac{||w||^2}{2} + C \sum_{i=1}^m \xi_i - \sum_{i=1}^m \alpha_i [y_i(x_i.w + b) + \xi_i - 1] - \sum_{i=1}^m \gamma_i \xi_i
\]

\( L_P \) should be minimized with respect to \( w, b, \xi_i \) and maximized with respect to \( \alpha_i \) and \( \gamma_i \). The Karuch-Kuhn-Tucker [17] for the primal problem are:

**Gradient Conditions**

\[
\frac{\partial L_P}{\partial w} = 0 \quad \Rightarrow \quad w = \sum_{i=1}^m \alpha_i y_i x_i
\]

\[
\frac{\partial L_P}{\partial b} = 0 \quad \Rightarrow \quad \sum_{i=1}^m \alpha_i y_i = 0
\]
\[ \frac{\partial L_P}{\partial \xi} = 0 \quad \Rightarrow \quad C = \alpha_i + \gamma_i \]

Orthogonality Condition \[ \alpha_i[y_i(w.x_i + b) - 1 + \xi_i] = 0, \quad \forall i \]

Feasibility Condition \[ y_i(w.x_i + b) - 1 + \xi_i \geq 0 \quad \forall i \]

Non-negativity Condition \[ \xi_i \geq 0, \quad \alpha_i \geq 0, \quad \gamma_i \geq 0, \quad \xi_i\gamma_i = 0, \quad \forall i \]

Substituting these equations into \( L_P \), we obtain the dual problem:

\[
\text{maximize} \quad L_D = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \\
\text{subject to} \quad 0 \leq \alpha_i \leq C \quad \forall \ i \quad \text{and} \quad \sum_{i=1}^{m} \alpha_i y_i = 0
\]

The solution for the vector \( w \) is \( w = \sum_{i=1}^{m} \alpha_i y_i x_i \)

From the KKT complementary slackness conditions we also have

\[ \alpha_i[y_i(w.x_i + b) - 1 + \xi_i] = 0 \]
\[ \xi_i\gamma_i = 0 \quad \Rightarrow \quad (C - \alpha_i)\gamma_i = 0 \]

Now \( \xi_i = 0 \) if \( \alpha_i < C \). Thus \( b \) can be obtained by taking average value for the patterns which have \( 0 < \alpha_i < C \). Finally to find the class \( y_k \) of any data point \( x_k \) for testing, we have the following formula:

\[
y_k = \text{sgn}(\sum_{i=1}^{m} \alpha_i y_i \langle x_i, x_k \rangle + b)
\]

2.1.3. Non Linear Two-Class SVM

Sometimes the data may not be separated by a linear hyperplane with best generalization in the original space because of its complex distribution. So we project the data by a real function \( \phi \) into a higher dimensional space, called feature space where it can be classified by a linear hyperplane as shown in figure 2.4. The function \( \phi : (x_1, x_2) \rightarrow (x_1^2, \sqrt{2}x_1 x_2, x_2^2) \) maps data from \( \mathbb{R}^2 \rightarrow \mathbb{R}^3 \) and can be separated by a linear hyperplane. This particular mapping has extremely useful property of allowing the computation of inner products of
feature vectors $\phi(x)$ and $\phi(w)$ by just squaring the inner product of the data vectors $x$ and $w$ in $\mathbb{R}^2$.

$$K(x, w) = (\langle x, w \rangle)^2 = (x_1w_1 + x_2w_2)^2 = x_1^2w_1^2 + 2x_1x_2w_1w_2 + x_2^2w_2^2$$

$$= x_1^2w_1^2 + (\sqrt{2}x_1x_2)(\sqrt{2}w_1w_2) + x_2^2w_2^2 = \langle (x_1^2, \sqrt{2}x_1x_2, x_2^2), (w_1^2, \sqrt{2}w_1w_2, w_2^2) \rangle = \langle \phi(x), \phi(w) \rangle$$

Thus mapping function $\phi$ fuses in kernel matrix $K$. Only the inner product of the data points are necessary and not the feature space coordinates. Kernel function is $K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$. $\phi$ is not required any more and it is possible to operate in any $n$-dimensional feature space without depending upon its complexity. This is called kernel trick and was invented by Vapnik et al. in 1992. Therefor if we simply replace $\langle x_i, x_j \rangle$ by $\langle \phi(x_i), \phi(x_j) \rangle = K(x_i, x_j)$, we can do non linear classification. Thus our objective function for non linear classifications becomes:

$$\text{maximize } L_D = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

subject to $0 \leq \alpha_i \leq C$ \forall \ i and $\sum_{i=1}^{m} \alpha_i y_i = 0$

We only need to find the kernel matrix $K$ whose $(i, j)$ entry will correspond to the inner product of feature space vectors $\langle \phi(x_i), \phi(x_j) \rangle$ without worrying about the images of $\phi(x_i)$ and
### Table 2.1. Few popular kernel functions

<table>
<thead>
<tr>
<th>Kernel Type</th>
<th>Formula</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Polynomial</td>
<td>((\alpha x^T y + c)^d)</td>
<td>Slope, alpha are adjustable</td>
</tr>
<tr>
<td>Gaussian</td>
<td>(exp(-|x - y|^2/2\sigma^2))</td>
<td>Decreasing sigma results in closer probing</td>
</tr>
<tr>
<td>Radial Basis</td>
<td>(exp(-|x - y|^2/2\sigma^2))</td>
<td>Also known as exponential kernel</td>
</tr>
<tr>
<td>Laplacian</td>
<td>(exp(-|x - y|/\sigma))</td>
<td>Same as RBF but less sensitive for (\sigma)</td>
</tr>
<tr>
<td>Sigmoid</td>
<td>(tanh(\alpha x^T y + c))</td>
<td>Used in Neural Networks, usually (\alpha = 1/(\text{Data Dimension}))</td>
</tr>
<tr>
<td>Log</td>
<td>(-log(|x - y|^d + 1))</td>
<td>Particularly interesting for images</td>
</tr>
</tbody>
</table>

\(\phi(x_j)\). Even though the kernel functions helps to classify quite complicated data distribution but we still have no mathematical theory to foresee what kernel and its parameters would give the optimal results. Sometimes we may have to repeat the experiments with different kernel functions to see which one suits the best. For example, the kernel which gives out too many support vectors is clearly not a good choice. The knowledge of data distribution also helps to choose the kernel function. We list few commonly used kernel functions in Table 2.1. Now we discuss one-class SVM which is also known as support vector data description (SVDD).

#### 2.2. One-Class SVM

It is useful for unbalanced data (one class is less than 10%) or the nature of negative class is unknown. For example while monitoring the state of a healthy person, we can collect many metrics or variables but when a person become sick, the causes are unknown. All we know is that sickness parameters do not match with healthy ones. So in one-class SVM we collect only the positive or salubrious class data for training and find optimal enclosing boundary for this class. During testing, the data which looks similar to the positive class is considered normal and everything else is negative or anomaly in its own way. There are two equivalent approaches for one-class SVM, separating data from the origin and minimum enclosing sphere.
Figure 2.5. Separating hyperplane and minimum enclosing sphere are equivalent [57] and called One-Class SVM

2.2.1. Separating Data From Origin and Minimum Enclosing Sphere

We solve a quadratic problem to obtain the decision function $F$ that takes the value +1 in a small region capturing most of the data and −1 elsewhere. Let’s start by separating data set from the origin approach. The test point $x_i$ is classified by $f(x)$ is determined by finding out which side of the hyperplane it falls in the feature space, shown in figure(2.5). We assume the target class to be positive and alien/anomaly class to be negative. The primal problem to separate the data from origin is

$$\begin{align*}
\text{minimize} & \; \frac{1}{2}\|w\|^2 - b + \frac{1}{\nu m} \sum_{i=1}^{m} \xi_i \\
\text{subject to} & \; (w.\phi(x_i)) \geq b - \xi_i, \; \xi_i \geq 0 \; \forall \; i
\end{align*}$$

where $w$ is a vector perpendicular to the hyperplane in the feature space, $b$ is the distance from the hyperplane to the origin, and $\xi_i$ are soft-margin slack variables to handle outliers. The parameter $\nu \in (0, 1)$ controls the trade-off between the number of records in the data set mapped as positive by the decision function $f(x) = sgn(\langle w.\phi(x) \rangle - b)$ and having a small value of $\|w\|$ to control model complexity. We use Lagrangian of equation (7) by using Lagrange multipliers $\alpha_i$ and $\beta_i$ to find the dual.

$$L_P = \frac{1}{2}\|w\|^2 - b + \frac{1}{\nu m} \sum_{i=1}^{m} \xi_i - \sum_{i=1}^{m} \alpha_i[\langle w.\phi(x) \rangle - b + \xi_i] - \sum_{i=1}^{m} \beta_i \xi_i$$
Setting the partial derivatives with respect to \((w, \xi_i, b)\) equal to zero we get.

\[
w = \sum_i \alpha_i \phi(x_i) \tag{9}
\]

\[
\alpha_i = \frac{1}{\nu l} - \beta_i \leq \frac{1}{\nu m} \sum_i \alpha_i = 1 \tag{10}
\]

Training data points with \(\alpha_i > 0\) are called Support Vectors and the decision function is

\[
f(x) = \text{sgn}(\sum_i \alpha_i K(x_i, x) - b) \tag{11}
\]

A newly collected data record \(x\) is predicted to belong to the target class if \(f(x) = 1\) and outlier class if \(f(x) = -1\). Thus our dual objective function now becomes

\[
\text{minimize } \frac{1}{2} \sum_{ij} \alpha_i \alpha_j K(x_i, x_j) \tag{12}
\]

subject to \(0 \leq \alpha_i \leq \frac{1}{\nu m}, \sum_i \alpha_i = 1\)

At the optimum, if \(0 < \alpha_i < \frac{1}{\nu m}\) then for such \(\alpha_i\) we have \(b = \langle w.\phi(x_i) \rangle = \sum_j \alpha_j K(x_j, x_i)\)

Now we consider the minimum enclosing ball problem in spirit of Support Vector Data Description (SVDD) [65] and see how it is equivalent to Equation(12). Here is the objective function to find that sphere with center \(c\) and radius \(R\) in the feature space:

\[
\text{minimize } R^2 + \frac{1}{\nu m} \sum_i \xi_i \tag{13}
\]

subject to \(\|\phi(x_i) - c\|^2 \leq R^2 + \xi_i, \xi_i \geq 0, \forall \ i\)

The dual objective function can be easily derived using Lagrangian and by replacing \(\phi(x_i).\phi(x_j)\) with Kernel dot product \(K(x_i, x_j)\) we get:

\[
\text{minimize } \sum_{ij} \alpha_i \alpha_j K(x_i, x_j) - \sum_i \alpha_i K(x_i, x_i) \tag{14}
\]

subject to \(0 \leq \alpha_i \leq \frac{1}{\nu m}, \sum_i \alpha_i = 1\)
For any support vector $x_k$ the center of the sphere is $c = \sum_k \alpha_k \phi(x_k)$ and the radius of the sphere can be calculated as $R^2 = \|\phi(x_k) - c\|^2$. Expanding the square using the inner product and the value of $c$ we get

\begin{equation}
R^2 = \sum_{i,j} \alpha_i \alpha_j K(x_i, x_j) - 2 \sum_i \alpha_i K(x_i, x_k) + K(x_k, x_k)
\end{equation}

If the data point lies within the feature space sphere then it would belong to the target class (+1), otherwise to the outlier class (−1). To test any point $z$, we have our classification function:

\begin{equation}
f(z) = \text{sgn} \left( R^2 - \sum_{ij} \alpha_i \alpha_j K(x_i, x_j) + 2 \sum_i \alpha_i K(x_i, z) - K(z, z) \right)
\end{equation}

Some kernels depend only on $x - y$, so for them $K(x, x)$ is constant and along with the summation of alpha, the linear term in Dual Equation (14) is constant. In that case we can simply ignore the second term and simplify the objective function. Now it becomes same Equation (12), the dual of 1-class SVM problem. Hence for constant $K(x, x)$ the feature space data lies on the sphere surface. Thus finding smallest enclosing sphere becomes same as finding smallest segment of the sphere where the data lies. This segment can also be found by simply intersecting the data sphere with the hyperplane. The hyperplane with maximum margin of separation from origin will cut off the smallest segment.
2.3. Two-Class SVM and One-Class SVM Comparison

The basic difference between two-class and one-class SVM is the usage of negative examples during training phase. Two-class SVM needs both positive and negative examples to learn the decision function and works great for balanced data. But negative examples may not be available in real world experiments such as newly deployed anomaly detection mechanisms since we do not know about failure stages and causes ahead of time. Neither do we know what types of anomalies could occur nor do we have anomaly history at the time of training. Therefore, until we have both classes of data available, we cannot apply two-class SVM. At the same time, this limitation of two-class SVM is the key attribute of one-class SVM. The one-class SVM requires a small training set of positive (normal class) examples only, to learn the decision function. As the working data set grows, it will eventually contain some abnormal records. As a result, two classes of data records will be available and two-class SVM will become a natural choice for anomaly detection since it is a powerful classification tool. It has been successfully applied to many applications. In our hybrid approach, we exploit best of both types of SVM algorithms to calculate the decision function. Our method also comprises the techniques for substantial spatial and computational savings while maintaining comparable accuracy.
In self-evolving anomaly detection mechanism, we have infinite stream of in-coming data and we want to detect malicious content in this data stream. For instance, consider HTTP server with in-coming requests and we are trying to find the virus or hacker attacks. From machine learning prospective there are several approaches to that, and support vector data description (SVDD and also known as one-class SVM) is also used for novelty/anomaly detection [65] which learns a concise description of normal data, i.e. model of normality by enclosing the normal data in a minimum enclosing sphere, and detects malicious code that deviates from this normality by lying outside of this sphere. But being fully unsupervised is its drawback for the practical problems. We always need to exploit the labeled data for high accuracy and better reliability of the detection results. For the self evolving systems we have infinite data streams and only few labels are available (normal and anomaly class). Thus SVDD has to be generalized to semi-supervised level [24] with the usage of active learning [60] to acquire the expert knowledge for the data close to the decision boundary (known as margin strategy) to reposition the sphere. Anomaly scores can also be assigned by calculating the distance of the new data point to the SVDD sphere center; less than $R$ means normal and anomaly otherwise.

3.1. Active and Semi-supervised Data Domain Description Algorithm

Goernitz et al. [24] proposed semi-supervised version of SVDD. Let $\{x_i, y_i\}_{i=1}^L$ and $\{x_i\}_{i=1}^U$ be the labeled and unlabeled data respectively. Suppose the normal data is positive class and anomalies belong to negative class. Using both labeled and unlabeled data for semi-supervised mechanism, the SVDD objective function would be

$$\minimize R^2 + C_L \sum_{i=1}^L \zeta_i + C_U \sum_{i=1}^U \xi_i - D\gamma$$

subject to $\|\phi(x_i) - c\|^2 \leq R^2 + \xi_i, \quad \xi_i \geq 0, \quad \forall \ i$

$y_i(\|\phi(x_i) - c\|^2 - R^2) \leq -\gamma + \zeta_i, \quad \zeta_i \geq 0, \quad \forall \ i$
where, $\gamma$ is margin offset, $D$, $C_U$ and $C_L$ are trade-off penalty parameters. Unfortunately the problem becomes non-convex after including negative labeled points so we eliminate the slack variables to make the problem unconstrained

$$\xi_i = \nu(R^2 - \|\phi(x_i) - c\|^2)$$

$$\zeta_i = \nu(y_i(\nu(R^2 - \|\phi(x_i) - c\|^2) - \gamma))$$

where $\nu(t) = \max(-t, 0)$ is called the hinge loss. The problem now becomes

$$\min R^2 + C_L \sum_{i=1}^{L} \nu(y_i(R^2 - \|\phi(x_i) - c\|^2) - \gamma) + C_U \sum_{i=1}^{U} \nu(R^2 - \|\phi(x_i) - c\|^2) \forall \ i$$

In order to make the function differentiable for conjugate gradient descent method, we replace hinge loss by Huber loss which is defined as follows:
\[ H(t) = \begin{cases} 
\Delta - t : t \leq \Delta - \epsilon \\
\frac{(\Delta + t - \epsilon)^2}{4\epsilon} : \Delta - \epsilon \leq t \leq \Delta + \epsilon \\
0 : t > \Delta + \epsilon 
\end{cases} \]

Taking the derivative of Huber loss function with respect to \( t \) we get

\[ H'(t) = \begin{cases} 
-1 : t \leq \Delta - \epsilon \\
-\frac{1}{2}(\Delta - t) + 1 : \Delta - \epsilon \leq t \leq \Delta + \epsilon \\
0 : t > \Delta + \epsilon 
\end{cases} \]

We further need to compute the gradients of slack variables \( \xi_i \) with respect to \( R \) and \( c \)

\[ \frac{\partial \xi_i}{\partial R} = 2RH'(R^2 - \|\phi(x_i) - c\|^2) \]
\[ \frac{\partial \xi_i}{\partial c} = 2(\phi(x_i) - c)H'(R^2 - \|\phi(x_i) - c\|^2) \]

Similarly gradients of \( \zeta_i \) with respect to \( R \), \( \gamma \) and \( c \) are

\[ \frac{\partial \zeta_i}{\partial R} = 2y_iRH'(y_i(R^2 - \|\phi(x_i) - c\|^2) - \gamma) \]
\[ \frac{\partial \zeta_i}{\partial \gamma} = -DH'(y_i(R^2 - \|\phi(x_i) - c\|^2) - \gamma) \]
\[ \frac{\partial \zeta_i}{\partial c} = 2y_i(\phi(x_i) - c)H'(y_i(R^2 - \|\phi(x_i) - c\|^2) - \gamma) \]

Solving the gradient of \( F = \text{Equation}(19) \) with respect to \( R \), \( \gamma \) and \( c \) after substitution

\[ \frac{\partial F}{\partial R} = 2R + C_U \sum_{i=1}^{U} \frac{\partial \xi_i}{\partial R} + C_L \sum_{i=1}^{L} \frac{\partial \zeta_i}{\partial R} \]
\[ \frac{\partial F}{\partial \gamma} = -D + C_L \sum_{i=1}^{L} \frac{\partial \zeta_i}{\partial \gamma} \]
\[ \frac{\partial F}{\partial c} = C_U \sum_{i=1}^{U} \frac{\partial \xi_i}{\partial c} + C_L \sum_{i=1}^{L} \frac{\partial \zeta_i}{\partial c} \]

Polack-Ribiere [16] flavour of conjugate gradients is used to compute the optimal solution of \( F \) i.e. Equation [19]. In case of non-trivial separating sphere, we can use Kernel feature
space by replacing the dot product of non-linear mapping functions $\phi(x_i)\cdot \phi(x_j)$ with kernel matrix element $K(i,j)$. By applying the Representer theorem \[58\] the center $c$ is

$$c = \sum_i \alpha_i \phi(x_i) + \sum_j \alpha_j y_j \phi(x_j) \tag{30}$$

Gradient of $F$ with respect to $\alpha_i/j$

$$\frac{\partial F}{\partial \alpha_k} = \frac{\partial F}{\partial c} \frac{\partial c}{\partial \alpha_{i/j}} \tag{31}$$

From Equations [30] and [31] we get

$$\frac{\partial c}{\partial \alpha_i} = \phi(x_i) \quad \text{and} \quad \frac{\partial c}{\partial \alpha_j} = y_j \phi(x_j) \tag{32}$$

Now we can write the gradient of $F$ with respect to $\alpha_{i/j}$. The final objective function along with the use of kernel functions can be stated as

$$\min R^2 - D\gamma + C_U \sum_{i=1}^U [H(R^2 - K(i,i) + (2e_i - \alpha)' K \alpha)] +$$

$$+ C_L \sum_{i=1}^L [H(y_i (R^2 - K(i,i) + (2e_i - \alpha)' K \alpha) - \gamma)] \tag{33}$$

where $e_i$ is the standard base elements i.e. 1 at $i$ position and zeros elsewhere. The semi-supervised SVDD constrained problem was not convex, so we translated it into unconstrained continuous problem of same numerical characteristics, which can be solved by gradient descent method. During the self evolving anomaly detection, SVDD solely starts with unlabeled data and then subsequent refinements happen by inclusion of labeled data. Labels can obtained either by the active learning process in which the the human experts are explicitly asked to label some important data points interactively or the labels can be provided voluntarily by the experts. The point $x_{user}$ is found using active learning strategy as follows and given to the user for its label.

$$x_{user} = \arg\min_{\{x_1,x_2,...,x_U\}} \lambda \frac{|f(x)|}{\Omega} + (1 - \lambda) \frac{1}{2k} \sum_{j=1}^{L+U} (\bar{y}_{ij} + 1)a_{ij} \tag{34}$$

$\lambda \in [0,1]$, $f(x) = |R^2 - \|\phi(x) - c\|^2|$ and $\Omega = \max_i |f(x_i)|$. The first term calculates the point closest to the decision hypersphere i.e. margin strategy. The second term uses $k$-nearest
neighbors to assign \( a_{ij} = 1 \) or 0 depending upon the adjacency of \( x_i \) and \( x_j \) and \( \bar{y}_i = y_i \) or 0 for unlabeled \( x_i \). This equation uses \( k \)-nearest neighbor graph and finds the point close to the decision boundary and lying in potential anomalous clusters. The user is then asked to label this point \( x_{user} \) for +1 or −1.

Thus in this algorithm, data domain description is viewed as SSL problem with a facility of expert knowledge through active learning. Empirical results in [24] show that the performance in network intrusion and object recognition tasks significantly improves the quality of learning using this SSL technique.

3.2. Semi-supervised Classification by Homogeneous Cluster Labeling

Support Vector Clustering (SVC) [4] can also be exploited with homogeneous cluster labeling for semi-supervised learning. SVC is based on SVDD (one-class SVM) and can detect quite irregular shaped clusters. It has three steps: compute support vector data description, compute adjacency matrix, compute connected components as the clusters and assign outliers. Adjacency matrix is composed of 1’s and 0’s, depending upon if two points lie in the same cluster or not. It needs space complexity of the order \( n \times n \) where \( n \) is number of data points. Moreover to check if two points belong to the same cluster or not, connected components computation and outliers assignments, are quite time consuming. Thus to avoid this bottleneck, we only use the first, out of three steps i.e. computing SVDD by solving the dual to find the minimum enclosing feature space sphere

\[
\text{minimize } \sum_{ij} \alpha_i \alpha_j K(x_i, x_j) - \sum_i \alpha_i K(x_i, x_i) \\
\text{subject to } 0 \leq \alpha_i \leq C, \sum_i \alpha_i = 1
\]

where \( K \) is Gaussian kernel matrix, since it works better than other known kernels for tight representation of clusters [65]. Solving the above quadratic problem gives us \( \alpha \) vector and based on that, we can divide our training data set into three partitions as shown in the table
Table 3.1. Definition of USV, BSV and NSV

<table>
<thead>
<tr>
<th>Points</th>
<th>Condition</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>USV</td>
<td>$0 &lt; \alpha_i &lt; C$</td>
<td>Margin or unbounded support vectors</td>
</tr>
<tr>
<td>BSV</td>
<td>$\alpha_i = C$</td>
<td>Bounded support vectors (errors)</td>
</tr>
<tr>
<td>NSV</td>
<td>$\alpha_i = 0$</td>
<td>Within a data description</td>
</tr>
</tbody>
</table>

The USV data lies on the surface of the sphere and decide its radius (its distance from the center). When this sphere is mapped back to the original data space, it forms a set of contours enclosing the data points and USVs lie on contours boundaries. These contours are interpreted as cluster boundaries and the enclosing data points in those contours form the clusters (figure 3.2). The Gaussian kernel parameter ($\sigma$) controls the probing level of the data and hence controls the number of clusters. Soft margin constant $C$ controls the smoothness of cluster boundaries. Decreasing its value reduces the decision boundary tension, reduces support vectors and increase outliers population.

To check if two points are in the same cluster or not, we consider the line segment joining those two points in the feature space and check if the entire line lies within the sphere or not. If yes, then the points belong to same cluster, otherwise not. This can be done by chopping the line into 20 points to see if each of their distance from the sphere center is less than $R$ or not. Here is the reason why this works. In feature space we look for the smallest sphere that encloses the image of the data. This sphere is mapped back to data space, where it forms a set of contours which enclose the data points. These contours are interpreted as cluster boundaries. Points enclosed by each separate contour are associated with the same cluster. figure (3.2) gives a nice visualization. Now, each contour level represents a sphere. If we are having our sphere at level $k$, to connect two clusters it would cut $(k + \delta)$ contour at least twice. As $(k + \delta)$ is a contour is outside $k$. Therefore, separate clusters can only be connected by an outside line.

Now, just for USVs (support vectors), we calculate adjacency matrix $A$ and cluster assignment phase, unlike SVC which does for all $n$ data points. Moreover according to the
available labeled data, we can have following three cases for making the homogeneous labeled clusters.

(1) If any labeled data (either positive or negative but not both) available in a cluster, then label all the support vectors lying on that cluster boundary with the same class label.

(2) If a cluster has no labeled data then we search for the nearest labeled cluster to use its label.

(3) If a cluster has both labels then we need to chop it into two, since making homogeneous labeled clusters means one cluster should have one class label.

Let us discuss each possibility separately.
Table 3.2. Comparison of time complexities between BB and HCL

<table>
<thead>
<tr>
<th>Number of Data Points</th>
<th>100</th>
<th>200</th>
<th>400</th>
<th>600</th>
<th>800</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch and Bound (sec)</td>
<td>0.78</td>
<td>1.6</td>
<td>7.6</td>
<td>19.2</td>
<td>51.68</td>
<td>194.53</td>
</tr>
<tr>
<td>Homogeneous Cluster Labeling (sec)</td>
<td>0.32</td>
<td>1.24</td>
<td>4.5</td>
<td>12.6</td>
<td>34.1</td>
<td>108.7</td>
</tr>
</tbody>
</table>

Figure 3.3. SVDD data description

Figure 3.4. Label support vectors

3.2.1. Non Overlapping Substantive Clusters (Case 1)

We are trying to label the outer walls of the clusters with appropriate labels to obtain the labeled data on the cluster boundaries. After we have labeled data on the cluster boundaries, SVM classification becomes a natural choice. This case has been illustrated in figure (3.3) - (3.6) using synthetic two moon dataset. Figure (3.3) contains 100 unlabeled data points (in two moons shape) with just two labeled points (one for each moon). SVDD calculates USVs which lie on the moon boundaries (circled dots) since each moon is a cluster. Figure (3.4) shows how the support vectors get homogeneously labeled. Figures (3.5) and (3.6) explain the usage of this labeled data for SVM classification. Table 3.2.1 is the comparison of the time complexities of homogeneous cluster labeling method and Branch and Bound [9] averaged over 10 random experiments.

Gaussian kernel because is the only kernel for which the number of maxima (hence the number of clusters) is a monotonically non-decreasing function of the parameter \( \frac{1}{2\sigma^2} \) [4]. Moreover this kernel produces a tight description in the kernel space and thus tight contours in the data space, while other kernel functions, e.g. the polynomial kernel stretches the data.
in the kernel space which causes the data to become very hard to describe in the hyperspace [65]. For consistence, we again use Gaussian kernel to calculate SVM plane for the obtained label data in figure (3.6). Fortunately, to select Gaussian kernel parameter \( \sigma \) for SVM, we have the following algorithm.

3.2.2. Automatic Method to Find Best RBF Kernel Parameter for SVM

C.H. Li et al. [37] proposed a method for automatic kernel parameter selection in SVM which we discuss here in brief. In the feature space determined by RBF (or Gaussian) kernel, the norm of each sample is positive and one, so the data is mapped onto the surface of a hypersphere. The cosine values (or Gaussian kernel function values) indicate similarities between data points; values close to 1 indicate similarity whereas close to 0 means dissimilarity in feature space. So the idea is that data from the same class should have mean of kernel values close to 1 and that of different class should be close to 0.

\[
\begin{align*}
  w(\sigma) &= \frac{1}{\sum_{i=1}^{L} N_i^2} \sum_{i=1}^{L} \sum_{j=1}^{N_i} \sum_{k=1}^{N_i} \exp(-\|x_i^j - x_i^k\|^2/2\sigma^2) \approx 1 \\
  b(\sigma) &= \frac{1}{\sum_{i=1}^{L} \sum_{j \neq i} N_i N_j} \sum_{i=1}^{L} \sum_{j \neq i} \sum_{l=1}^{N_i} \sum_{k=1}^{N_j} \exp(-\|x_i^l - x_j^k\|^2/2\sigma^2) \approx 0
\end{align*}
\]

Minimize \( J(\sigma) = 1 - w(\sigma) + b(\sigma) \)
The objective is to minimize \( J(\sigma) \) which has global minimum on \((0, \infty)\). Since \( w(\sigma) \) and \( b(\sigma) \) are differentiable with respect to \( \sigma \), so we can use Gradient Descent method to find the minimizer for \( J(\sigma) \).

\[
\sigma_{n+1} = \sigma_n - \gamma_n \nabla J(\sigma_n) \tag{39}
\]

\[
\nabla J(\sigma_n) = \frac{\partial b(\sigma_n)}{\partial \sigma} - \frac{\partial w(\sigma_n)}{\partial \sigma} \tag{40}
\]

A graph “\( \sigma \) versus \( J(\sigma) \)” can also be drawn to see the minima, which is approximate to achieve the highest accuracy. This procedure is much faster than \( k \)-fold cross validation to find the proper kernel parameter for SVM classification. In figure (3.7) we can see that the optimal kernel value is \( \sigma = 0.3 \). SVM plane corresponding to this \( \sigma \) and the labeled data is shown in figure (3.6). Now back to our SSL learning strategy for case (1). Sometimes we could have overlapping clusters, then support vector clustering is still useful but with different interpretation. By decreasing the value of soft margin parameter \( C \) we allow more BSVs and interpret the feature space to figure out cores of the clusters instead of enclosing most of the data. For example figure (3.8) shows first two PCA components of IRIS flower dataset. It has 4 dimensions and 3 samples of 50 data points each \((150 \times 4)\). Among three classes, the second and third class is highly overlapping, so we adjust \( C \) and \( \sigma \) for SVDD to find the major clusters. The cluster frequency histogram in figure (3.10) confirms that we have 3 major clusters and many small clusters. We only need three labels for these three clusters and can get their respective boundary USVs labeled similar to the two moon
3.2.3. Non Overlapping Grouped Clusters (Case 2)

If a cluster has no label then k-nearest neighbor algorithm can be used to obtain the label for a dry cluster. So big clusters share their labels with smaller nearby unlabeled clusters.

3.2.4. Overlapping Clusters (Case 3)

Problem of case (3) is complicated, i.e. when a cluster has both class labels and we need to chop it, while considering the data distribution into mind. This occurs into highly overlapping clusters where the smoothness assumption (data points in high density regions share the same label) fails. To overcome this problem Zhang et al [72] introduced a subspace regularization method for SSL.

**Subspace Regularization Semi-Supervised Learning Algorithm:** This algorithm [72] makes the classic low-dimensionality embedding assumption, i.e. most high dimensional data information lies in some low dimensional manifold. For example, diverse face expression pictures of a man are a result of few muscle movements. Thus the problem is two folds, finding a subspace and a decision function on that subspace which separates the data while the geometric information is maximally preserved. This is an iterative algorithm based on traditional projection pursuit procedure [30] with low computational complexity and used
for large scale data sets. Given the training set \( X = X_L \cup X_U \), where \( X_L \) and \( X_U \) are labeled and unlabeled data respectively such that \( X_L = \{(x_i, y_i)\}_{i=1}^L \), \( X_U = \{x_i\}_{i=1}^U \) and \( L \ll U \). Let \( W \) be the subspace with dimension \( P \) where we need the decision function \( g \) minimizing the following objective function:

\[
L = \sum_{i=1}^L \Delta_g(y_i, g(x_i^T W)) + \lambda \Delta_W(X, X^W)
\]

where \( \Delta_g \) and \( \Delta_W \) are loss functions corresponding to the decision function \( g \) and information loss by projecting the data in subspace \( W \). By choosing \( \Delta_g \) as least square error, \( \Delta_W \) as reconstruction error and \( g \) be arbitrary linear function in \( W \) we get:

\[
L = \sum_{i=1}^L \left( y_i - \sum_{k=1}^P (\alpha_k x_k^T w_k) \right)^2 + \lambda \sum_{i=1}^{L+U} \|x_i - x_i^W\|^2
\]

We can iteratively solve this equation to find \( \alpha \) and \( W \) whereas \( P \) and \( \lambda \) are user defined hyper-parameters which can be chosen by cross-validation. By increasing \( \lambda \) the reconstruction error \( \Delta_W \) gets more preference over the data-fitting error \( \Delta_g \) and the subspace tends to get closer to PCA subspace. The name of the algorithm is PCA-based least square algorithm and in each iteration, one projection direction is added into the subspace followed by finding \( g \) such that the \( L \) is reduced as much as possible.

Consider the \( t \)th iteration when \( W \) has \( t-1 \) dimensions. Suppose \( R^g \) and \( R^W \) are the residues for decision function \( g \) and for subspace \( W \) respectively.

\[
g(v) = \sum_{j=1}^{t-1} \alpha_j v_j
\]

\[
R_i^g = y_i - \sum_{j=1}^{t-1} \alpha_j x_j^T w_j \quad \text{(} y_i \text{ is decision response of } g \text{)}
\]

\[
R_i^W = x_i - x_i^W = x_i - \sum_{j=1}^{t-1} \beta_j w_j \quad \text{(} R_i^W \text{ is } \perp \text{ to } W \text{)}
\]

For \( t \)th iteration, the objective function becomes

\[
\min_{\alpha, \beta, w} F = \sum_{i=1}^L \left( y_i - \sum_{j=1}^{t-1} \alpha_j x_j^T w_j - \alpha x_i^T w \right)^2 + \lambda \sum_{i=1}^{L+U} \|x_i - \sum_{j=1}^{t-1} \beta_j w_j - \beta_i w\|^2
\]
\[
\sum_{i=1}^{L} (R^g_i - \alpha x_i^T w)^2 + \lambda \sum_{i=1}^{L+U} \|R^W_i - \beta_i w\|^2
\]

(47) \quad = \|R^g - \alpha X^L w\|^2 + \lambda \sum_{i=1}^{L+U} \|R^W_i - \beta_i w\|^2 \quad \text{such that } \{w \perp w_j\}_{j=1}^{L-1}

where \(X^L\) refers to first \(L\) columns of data matrix \(X\), \(X^L^T\) is the transpose of \(X^L\), \(\alpha\) and \(\beta\) are scalars. \(w_t\) is added to \(W\) after this iteration i.e. a new subspace dimension is added.

The problem can be simplified by limiting the search to the subspace spanned by residuals i.e. \(w = \sum_{i=1}^{L+U} \gamma_i R^W_i = R^W \gamma\). In [72] it is proved that the optimal \(w^*\) is essentially a linear combination of \(R^W_i\). This eliminates \(w\) from the objective function and it becomes unconstrained

\[
F = \|R^g - \alpha X^L R^W \gamma\|^2 + \lambda \sum_{i=1}^{L+U} \|R^W_i - \beta_i R^W \gamma\|^2
\]

(48) This objective function is solved by coordinate-descent iterative method. In each step we optimize \(\alpha, \beta\) for fixed \(\gamma\) then optimize \(\gamma\) for fixed \(\alpha, \beta\). Let us fix \(\gamma\) constant and make partial derivatives of \(F\) with respect to \(\alpha\) and \(\beta\) zero.

\[
\frac{\partial F}{\partial \alpha} = 0 \quad \Rightarrow \quad \alpha = \frac{\langle R^g, X^L R^W \gamma \rangle}{\langle X^L R^W \gamma, X^L R^W \gamma \rangle}
\]

(49) \quad \frac{\partial F}{\partial \beta} = 0 \quad \Rightarrow \quad \beta = \frac{\langle R^W_i, R^W \gamma \rangle}{\langle R^W \gamma, R^W \gamma \rangle}

(50) These values are used to update \(\gamma\). Now \(\alpha\) and \(\beta\) are held constant and partial derivative of \(F\) is obtained:

\[
\frac{\partial F}{\partial \gamma} = -2\alpha R^W X^L R^g + 2\alpha^2 R^W X^L X^L R^W \gamma + 2\lambda R^W R^W [\left(\sum_{i=1}^{L+U} \beta_i^2\right) \gamma - \beta]
\]

(51) Iterative coordinate descent method gives the optimal solution \((\alpha^*, \beta^*, \gamma^*)\) followed by updating \(W\), \(R^g\) and \(R^W_i\) (where \(R^W_i \perp W\) makes it identical to Gram-Schmidt orthogonalization).

It takes \(p\) iterations to get the best subspace and the decision function.

If the data is not linearly separable then the kernel trick can be applied and the above linear case can be converted into non linear setting. Simply replacing \(R^W_i = \phi(x) M_i\) and \(K(i,j) = \langle \phi(x_i), \phi(x_j) \rangle\) does the job. Here \(\phi(X) = \{\phi(x_i)\}_{i=1}^{L+U}\) maps the data points to the
feature space, and $M$ is square matrix of the order $(L + U)$ containing real coefficients. The computation complexity of the algorithm is $O(PN D)$, where $P$ is dimension of subspace, $N$ is number of input data points and $D$ is the dimensions of input data.

Subspace regularization algorithm works remarkably well even when the data classes overlap heavily, avoids the curse of dimensionality and reduces the complexity of the problem. This method works much better than three other smoothness based methods, Gaussian random field [74], learning with local and global consistency [73], and manifold regularization [3].

3.3. Semi-Supervised Adaptive Anomaly Detection (AAD)

Anomaly or outliers detection refers to detecting the abnormal behaving pattern(s) in a given dataset. Anomaly detection has numerous real life applications such as fault detection in a network system, fraud detection for credit cards, cyber-security intrusion, system health monitoring for safety and experimental errors/noise discovery. In real life applications, data comes in a streaming manner, thus requires on-line data analysis. We have to maximize the detection sensitivity by keeping low false alarm rate because for large data sets even a small percentage false alarm can overload the expert’s work to examine the system’s health. Autonomic anomaly detection is a crucial technique for understanding emergent, self-managing and system-level dependability assurance. To detect anomalies, we need to monitor the system execution and collect runtime performance data. These data are usually unlabeled, and thus a prior failure history is not always available, especially for newly managed or deployed systems.

In this section, we present an adaptive anomaly detection (AAD) framework for dependability assurance. It employs data description using hypersphere to detect anomalies. Without loss of generality, we assume the given AAD system is newly deployed or managed, i.e. no history information of anomaly occurrences is available. The AAD detector identifies anomalies by searching for those data set states that are significantly different from the

Section 3.3 was accepted for publication and is presented in its entirety at [45] Brief idea of this section also was accepted as poster paper in [46].
Figure 3.11. Minimum enclosing sphere illustration in the Gaussian kernel feature space (circles represent anomalies and crosses are normal data points)

majority. As the detections get verified by the data set operators, they are confirmed as either true anomalies or normal states (false alarms). Our AAD anomaly detector adapts itself by recursively learning from these newly verified detection results to refine future detections. Meanwhile, the data set operators report observed but undetected anomaly records to the AAD detector, which exploits these records to identify new types of anomalies in the future. The health performance data set are continuously collected at runtime. Initially, all the data records are treated as normal ones. As time goes by, a small percentage of anomaly records is detected. Each of those detections is then verified and labeled as either a normal event or a anomaly with the corresponding anomaly type. For experimental elucidation, we have implemented a prototype of the our algorithm and conducted experiments in an on-campus cloud computing environment. Our experimental results show that AAD can achieve more efficient and accurate anomaly detection than other existing schemes.

3.3.1. The Objective Function

We explore support vector data description (SVDD) to describe the incoming performance data, detect anomalies, and adapt the anomaly detector when verified detection results are available. To start with the normal system performance data description, we define a hypersphere which gives a closed boundary around the data points with center \( o \) and radius \( R > 0 \). We minimize the volume of the hypersphere by minimizing the radius, and
demand that the hypersphere contain as many performance data points \( \{x_i\} \) as possible.

\[
\min L = \sum_i \sum_j \alpha_i \alpha_j K(x_i, x_j),
\]

subject to \( \sum_i \alpha_i = 1 \) and \( 0 \leq \alpha_i \leq A, \ i = 1, 2, \ldots \)

were \( A \) is the soft margin parameter, \( K(x_i, x_j) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}} \) following the Gaussian kernel and \( \sigma \) is the kernel parameter. Minimizing \( L \) gives a set \( \alpha_i \). A performance data point \( x_i \) satisfies one of the following inequalities:

\[
\begin{align*}
\|x_i - o\|^2 < R^2 &\implies \alpha_i = 0, \ \gamma_i = 0 \implies \text{Inside the hypersphere (NSV)} \\
\|x_i - o\|^2 = R^2 &\implies 0 < \alpha_i < A, \ \gamma_i = 0 \implies \text{On the hypersphere (USV)} \\
\|x_i - o\|^2 > R^2 &\implies \alpha_i = A, \ \gamma_i > 0 \implies \text{Outside the hypersphere (BSV)}
\end{align*}
\]

Center and radius of the hypersphere in kernel space are determined by the following equations (\( i \) is the index of support vectors)

\[
\begin{align*}
o &= \sum_i \alpha_i \phi(x_i), \\
R^2 &= \|\phi(x_i) - o\|^2 \quad \text{(for any support vector } x_i) \\
&= K(x_i, x_i) - 2 \sum_j \alpha_j K(x_j, x_i) + \sum_j \sum_l \alpha_j \alpha_l K(x_j, x_l)
\end{align*}
\]

3.3.2. AAD Algorithm

The adaptive anomaly detection (AAD) mechanism works as follows. Initially when no prior anomaly records are available, the performance data are unlabeled, AAD detector constructs a hypersphere to cover the majority of data records, by solving the dual problem in Equation (52). After mapping the hypersphere to the data space, those data points that lie outside the contours are identified as possible anomalies. Then, they are reported to the data analysts, who verify and confirm those detections as either true anomalies or normal states. The AAD detector learns from the verification results and updates the SVs of the hypersphere, and thus its center and radius using Equation (54). The data analysts also periodically report the observed but undetected anomaly events, which are explored by the AAD detector to adapt the hypersphere. For newly collected performance data records, the
AAD detector employs the updated hypersphere to identify possible anomalies. Algorithm 1 presents this adaptive anomaly detection process.

---

**Algorithm 1. Adaptive anomaly detection**

\[\text{AADanomalyDetection}()\] {

1: \( X = \) performance dataset;
2: \( q = \frac{1}{2\sigma^2}; \) \hspace{1cm} //Initialize kernel width
3: \( A = \frac{1}{n} + 10^{-3}; \) \hspace{1cm} //Initialize A slightly bigger than 1/n
4: \( \alpha = \text{solution to Dual}(X, q, A); \) \hspace{1cm} //Equation (52)
5: \( o = \sum_i \alpha_i \phi(x_i); \) \hspace{1cm} //Center of the hypersphere
6: \( R^2 = \|\phi(x_i) - o\|^2; \) \hspace{1cm} //Radius of the hypersphere
7: \textbf{while} (TRUE) \textbf{do}
8: \hspace{1cm} On receipt of a performance data record \( x_i \)
9: \hspace{1.5cm} \textbf{if} \( \|\phi(x_i) - o\|^2 > R^2 \) \textbf{then}
10: \hspace{2.5cm} report a possible anomaly with performance states \( x_i; \)
11: \hspace{1.5cm} \textbf{end if}
12: \hspace{1cm} On receipt of a verified detection or an observed but undetected anomaly \( f_j \)
13: \hspace{1.5cm} \textbf{if} \ (f_j \text{ is normal AND } \|\phi(f_i) - o\|^2 > R^2) \text{ OR }
14: \hspace{2.5cm} (f_j \text{ is a anomaly AND } \|\phi(f_i) - o\|^2 < R^2) \text{ then}
15: \hspace{2.5cm} \quad q = q + \delta; \quad //Adapt q
16: \hspace{2.5cm} \quad A = A + \Delta; \quad //Adapt A
17: \hspace{2.5cm} \quad \alpha = \text{solution to Dual}(X, q, A);
18: \hspace{2.5cm} \quad \text{update the center } o \text{ and radius } R;
19: \hspace{1.5cm} \textbf{end if}
20: \textbf{end while}
In Algorithm 1, the values of \( q \) and \( A \) are updated by \( \delta \) and \( \Delta \) respectively to adapt the hypersphere (Lines 14 and 15). This makes the updated hypersphere covers most of the available normal performance data points. The values of \( \delta \) and \( \Delta \) are tuned at runtime to achieve a high ROC slope for anomaly detection.

3.4. Hybrid Anomaly Detection (HAD)

Our proposed self-evolving and hybrid anomaly detection framework includes two components. One is detector determination. The detector is self-evolving and constantly learning. For a newly collected data record, the detector will calculate an abnormality score. If the score is below a threshold, a warning will be triggered, possibly with the type of abnormality which may help a system administrator to pin point the anomaly. The other component is detector retraining and working data set selection. The detector needs to be retrained when certain new data records are included in the working data set. In addition, working data set selection is imperative since the size of available health-related data from large-scale production systems may easily reach hundreds and even thousands giga-bytes. The detector can not blindly use all available data. For high dimensional data sets, we may need metric selection and extraction which work in a horizontal fashion while working data selection is vertical or sequential. Clearly, all these components are important and they will be orchestrated to achieve accurate and efficient real time anomaly detection.

Again without loss of generality, we assume the given system is newly deployed or managed. Health-related system status data, such as system logs, will be gradually collected. The size of the data set will quickly grow from zero to something very large. Initially, all the data records are normal. As time goes by, a small percentage of abnormal records will appear. Those abnormal records can be labeled according to their anomaly types. The

---

Section 3.4 was accepted for publication and is presented in its entirety at [40] with Springer publication. Some parts of this section are also included in accepted paper [41].

36
detector will be a function generated by the one-class SVM. To be more specific, let \( D \) be the working data set including \( m \) records \( x_i \in \mathbb{R}^d \) \((i = 1, 2, ..., m)\). Let \( \phi \) be a mapping from \( \mathbb{R}^d \) to a high dimensional feature space where dot products can be evaluated by some simple kernel functions:

\[
k(x, y) = \langle \phi(x), \phi(y) \rangle
\]

A common kernel function is the Gaussian kernel \( k(x, y) = -\frac{\|x-y\|^2}{2\sigma^2} \). The idea of one-class SVM is to separate the data set from the origin by solving a minimization problem:

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

subject to \( (w.\phi(x_i)) \geq b - \xi_i, \quad \xi_i \geq 0 \quad \forall \ i \)

where \( w \) is a vector perpendicular to the hyperplane in the feature space, \( b \) is the distance from the hyperplane to the origin, and \( \xi_i \) are soft-margin slack variables to handle outliers. The parameter \( \nu \in (0, 1) \) controls the trade-off between the number of records in the data set mapped as positive by the decision function \( f(x) = \text{sgn}((w.\phi(x_i)) - b) \) and having a small value of \( \|w\| \) to control model complexity. In practice, the dual form of (56) is often solved. Let \( \alpha_i \) \((i = 1, 2, ..., m)\) be the dual variables. Then the decision function is \( f(x) = \text{sgn}((w.\phi(x_i)) - b) \). A newly collected data record \( x \) is predicted to be normal if \( f(x) = 1 \) and abnormal if \( f(x) = -1 \). One of the advantages of the dual form is that the decision function can be evaluated by using the simple kernel function instead of the expensive inner product in the feature space. As the working data set grows, it will eventually contain some abnormal records. In other words, two classes or multiple classes of data records will be available. Therefore, SVM will become a natural choice for anomaly detection since SVM is a powerful classification tool and has been successfully applied to many applications.

The soft-margin binary SVM is similar to the above equation can be formulated using the slack variables \( \xi_i \) :

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

subject to \( y_i((w.\phi(x_i)) + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \quad \forall \ i \)

37
where $C > 0$ is a parameter to deal with misclassification and $y_i \in \{+1, -1\}$ are given class labels. A data is solved and the decision function is $f(x) = \text{sgn}(\sum_i \alpha_i k(x_i, x) + b)$. A newly collected data record $x$ could be predicted to be normal if $f(x) = 1$ and abnormal if $f(x) = -1$. Multi-class classification can be done using binary classification.

3.4.1. Detector Determination

A challenge to SVM is that the working data set is often highly unbalanced: normal data records outnumber abnormal data records by big margin. Classification accuracy of SVM is often degraded when applied to unbalanced data sets. However, as the percentage of abnormal data records increases, the performance of SVM will improve. Our numerical experiments show that SVM starts to perform reasonably well for this particular unbalanced problem once the percentage reaches 10%. Our detector is determined by combining one-class SVM and SVM with a sliding scale weighting strategy. This strategy can easily be extended to including other classification methods.

The weighting is based on two factors. One is credibility score and the other is the percentage of abnormal data records in the working data set. The method with a higher credibility score will weigh more and more weight will be given to SVM as the percentage of abnormal data records increases. For a given method, let $a(t)$ denote the numbers of attempted predictions and $c(t)$ denote the number of correct predictions where $t$ is any given time. The credibility score is defined to be

$$s(t) = \begin{cases} \frac{c(t)}{a(t)} & \text{if } a(t) > 0 \text{ and } \frac{c(t)}{a(t)} > \lambda \\ 0 & \text{if } a(t) = 0 \text{ or } \frac{c(t)}{a(t)} \leq \lambda \end{cases}$$

where $\lambda \in (0, 1)$ is a parameter of zero trust. A good choice is $\lambda = 0.5$. Let $s_1(t)$ and $s_2(t)$ be the credibility scores of one-class SVM and SVM, respectively. Let $p(t)$ denote the percentage of abnormal data records in the working data set. Suppose $f_1(x)$ is the decision function generated by one-class SVM and $f_2(x)$ is generated by SVM where $x$ is a newly
collected data record at time $t$. Then the combined decision function is given by

$$f(x) = \begin{cases} f_1(x)s_1(t) & \text{if } p(t) = 0 \\ \frac{1}{2}(f_1(x)s_1(t) + f_2(x)s_2(t)) & \text{if } p(t) \geq \theta \\ f_1(x)s_1(t)(1 - \frac{p(t)}{2\theta}) + f_2(x)s_2(t)(\frac{p(t)}{2\theta}) & \text{if } 0 < p(t) < \theta \end{cases}$$

(58)

where $\theta \in (0, 1)$ is a parameter of trust on SVM related to the percentage of abnormal data records. A reasonable choice is $\theta = 0.1$. An anomaly warning is triggered if $f(x)$ is smaller than a threshold $\tau$, say, $\tau = 0$. When multiple labels are available for abnormal data records, a multi-class SVM can be trained to predict the type of anomaly if a new data record is abnormal.

### 3.4.2. Detector Retraining and Working Data Set Selection

Detector retraining and working data set selection are part of a learning process. The basic idea is to learn and improve from mistakes and maintain a reasonable size of the data set for efficient retraining. Initially, all data records are included in the working data set to build up a good base to train the detector. Once the data set reaches a certain size and the detection accuracy is stabilized, the inclusion will be selective. A new data record $x$ is included in the working data set only if one or more of the following is true:

- The data record corresponds to an anomaly and $p(t) < 0.5$. It is ideal to include more abnormal data records in the working data set but not too many.
- One of the predictions by $f_1(x)$, $f_2(x)$, or $f(x)$ is incorrect. The detector will be retrained to learn from the mistake.
- The data record may change the support vectors for SVM. This happens when the absolute value of $\sum_i \alpha_i k(x_i, x) + b)$ is less than 1, where we assume $f_2(x) = \text{sgn}(\sum_i \alpha_i k(x_i, x) + b)$. The detector will be adjusted to have better detection accuracy.

The decision functions $f_1(x)$ and $f_2(x)$ will be retrained whenever a new data record enters the working data set. The retraining can be done quickly since the size of the data set is well maintained. In addition, the solutions of the old one-class SVM and SVM can be used as
the initial guesses for the solutions of the new problems. Solving one-class SVM and SVM is an iterative process. Having good initial guesses will make the iterations converge fast to the new solutions.

3.4.3. Sample Margin Information For Updating Working Dataset

To update the working dataset, trained data are partitioned into three categories based on KKT conditions, USV, BSV and NSV. The computational complexity of our anomaly detection method is proportional to the size of dataset window so the increment of data size causes scale problems in detector retraining. The spatial complexity is even more serious because all trained data have to be preserved. To make detector retraining more scalable in a real large problems, we need to remove useless data. In our approach, we exploit complexity reduction method by removing useless data based on the sample margin [33].

Detector retraining of our anomaly detection algorithm is a method finding a new decision boundary considering only data trained up to present. Because all data are not trained, the current data description is not optimal for whole dataset but it can be considered as an optimal data description for trained data up to now. We can eliminate every NSVs classified by the current hyperplane. However it is risky because important data which have a chance to be unbounded support vectors (USVs) might be removed as learning proceeds incrementally so the current hyperplane may not converge on the optimal hyperplane.

Therefore we need to define cautiously removable NSVs using sample margin. To handle the problem of removing data which become USVs, we choose data whose sample margin is in the specific range as removable NSVs. As shown in figure 3.12, we intend to select data in the region above the gray zone as removable NSVs. The gray region called epsilon region. It is defined to preserve data which may become USVs. The removable NSV is defined as follows:

**Definition 3.1: Candidate of removable NSV** The data $x$ that satisfies the following con-
dation should be removed from the dataset window.

\[ \gamma(x) - \gamma(USV) \geq \epsilon(\gamma_{\text{max}} - \gamma(USV)) \]  

where \( \epsilon \in (0, 1] \) is the user defined coefficient, \( \gamma(USV) \) is the sample margin of support vectors which is on the boundary and \( \gamma_{\text{max}} = \max_{i \in NSV} \gamma(x) \).

As in figure 3.12, by preserving data in \( \epsilon \) region, an incremental detector retraining using sample margin information can obtain the same data description as original incremental anomaly detector with less computational and spatial load. If \( \epsilon = 0 \) is, then we assume all data lying on the upper side of hyperplane as the candidates of removable non support vectors, and this makes learning unstable. When \( \epsilon = 1 \), we can hardly select removable NSVs, so the effect of speeding up and storage reduction is meager.

The performance data may be very high dimensional and clustering faces curse of dimensionality according to Steinbach et al [62]. Problems with high dimensionality happens because a given number of points become sparser when we increase the dimensions. Suppose we have 100 points in the interval \([0, 1]\) and from uniform random distribution. If we break \([0, 1]\) into 10 pieces then it is highly probable that each piece would contain some points. Now suppose we distribute the same number of points on a unit square, then the probability of each piece of size \((0.1)^2\) to contain some points would decrease. If we further increase the dimensions to three by considering a unit cube then each piece of size \((0.1)^3\) would have very little chance to contain a point because now we have 1000 small piece cubes and only
100 points distributed among them, so most of the pieces would be empty. Hence the data becomes more sparse by increasing the dimensions.

The collected health performance data in our algorithm is very high dimensional and we need to intensify the data by reducing its dimensions by using ICA. Independent component analysis (ICA) is a recently developed method by Hyvarinen [31] in which the goal is to find a linear representation of nongaussian data so that the components are statistically independent, or as independent as possible. It is used for feature extraction and signal separation of the data to apprehend its substantial pattern. Now we discuss our HAD algorithm for self-evolving anomaly detection.

3.4.4. HAD Algorithm

Algorithm 2. : Hybrid Anomaly Detection

```plaintext
HybridAnomalyDetection() {
1: X = Initialize working dataset for training;
2: Get ICAcoeffmatrix(X);
3: Y = Initialize labels; // normal=1, anomaly =-1, unknown=0
4: Tran1and2classSVM(X,Y);
5: while(TRUE) {
6:   GetNewDataPoint(x) ; // one receipt of performance data x
7:   x = ICAcoeffmatrix*x; // obtain ICA components of x
8:   Calculate s1(t) and s2(t) ; // credibility scores of one and two class SVMs
9:   Calculate f1(t) and f2(t) ; // decision functions of one and two class SVMs
10:  Calculate f(x) ; // hybrid decision function of one and two class SVMs
11:  X = DetectorRetrain(X,x);
12:  Calculate p(t)
13:  end while
```
X = DetectorRetrain(X,x) {

1: if x is anomaly and p(t) < 0.5
2: Include x into working dataset X
3: Tran1and2classSVM(X,Y); // retrain SVDD and SVM
4: return;
5: end if

6: if prediction by either f_1(x), f_2(x), f(x) is incorrect OR
7: |\sum_i a_i k(x_i, x) + b| < 1 for SVM
8: Include record x in working dataset X
9: Tran1and2classSVM(X,Y);
10: Resize(X); //using sample margin information
11: return;
12: end if
13: }

Predefine MAXSIZE (of working dataset) and \( \epsilon \) (Defintion 3.1)

Resize(X) {

1: if Sizeof(normal class) or Sizeof(anomaly class) > MAXSIZE
2: Find removable NSV using definition (59) with a given \( \epsilon \) value
3: Remove NSV from Dataset X
4: end if
5: }

Tran1and2classSVM(X,Y) {
1: Calculate $p(t)$;
2: \textbf{if} $p_t<0.1$
3: \hspace{1em} TrainSVDDonly(X,Y);
4: \textbf{else}
5: \hspace{1em} TrainSVMandSVDD(X,Y);
6: \textbf{end if}
7: \}

Thus our self evolving semi-supervised anomaly detector identifies possible anomalies in the collected performance data. It adapts itself by learning the verified detection results and observed but undetected failure events reported from the data analysts. In next chapters we employ our algorithms to cloud computing infrastructure as an application and see the experimental performance. But our algorithms is general purpose and can be applied for any large and streaming data set to detect outliers in a similar way.
CHAPTER 4

INTRODUCTION TO CLOUD COMPUTING

Cloud computing is an environment in which, (i) applications are delivered as services over the Internet and (ii) hardware and systems software in the data centers provide those services [1]. The cloud refers to the data center hardware and software and the services are known as software as a service (SaaS). A public cloud is developed when a cloud is available in a pay-as-you-go manner to the general public. On the other hand private cloud is created when internal data centers of a business or other organization are not made available to the general public. The service being sold is known as utility computing. Thus, SaaS + utility computing = cloud computing, not including private clouds. People have the option of being

This chapter is written for self containment of our research. Selected references are [1], [2] and [10].

Figure 4.1. Cloud computing infrastructure [10]
Presently, cloud computing is still a changing prototype[2]. The definitions, attributes, and characteristics will continue to change and be redefined over time with the continued use by public and private sectors. This definition attempts to encompass the cloud approaches in an institute wide cloud computing system with our anomaly detection mechanism.

4.1. Definition

Cloud computing is a paradigm that allows easy, on-demand network access to a shared pool of configurable computing resources. These computing resources consists of networks, servers, storage, applications, and services. They require little managerial effort or host interaction therefore, can be rapidly provided and discharged. The cloud model consists of five essential characteristics, three service models, and four deployment models. The main function of cloud model is to enhance accessibility.

4.2. Essential Characteristics

1) **On-demand self-service** A consumer has the capability to automatically and independently provide the computing abilities as desired e.g. network storage and server time since each service provider does not require human interaction.
(2) **Broad Network Access:** Capabilities are primarily accessible on the network. The standard mechanisms that promote use by diverse thick or thin client platforms, such as mobile phones, laptops, and PDAs, allow capabilities to be utilized.

(3) **Resource Pooling:** The provider uses a multi-guest model to merge the computing resources, with different physical and virtual appliances dynamically assigned/reassigned according to consumer demand and serve multiple clients. The client usually has no idea about the location of the provided resources but could be able to choose a location e.g., country, state, or data center. The common shared resources are storage, processing, memory, network bandwidth, and virtual machines.

(4) **Rapid Elasticity:** Facilities can be rapidly, flexibly and automatically supplied or released in order to quickly scale in and out. Client can buy any amount of facilities any time which seem unlimited to the client.

(5) **Measured Service:** By influencing a metering capability to the particular service type at some level of remoteness, the cloud systems automatically control and optimize the resource. These services could be storage, processing, bandwidth and active user accounts. In order to put the host and the client on the same page for the service utilization, the resource usages can be monitored, controlled and reported to both of them.

**Definition (Cloud Infrastructure):** It is the collection of hardware and software that enables the five essential characteristics of cloud computing and contains a physical layer and an abstraction layer. The physical layer consists of necessary hardware resources such as servers, network components, storage and the abstraction layer consists of the software in the physical layer that demonstrates the basic cloud attributes.

4.3. Service Models

There are three types of cloud service models, a software provider, a computing platform provider and the most basic infrastructure service provider for computers as physical or virtual machines.
(1) **Cloud software as a service (SaaS):** Cloud providers install and operate software in the cloud infrastructure and clients get the access. The cloud users can access the applications through a thin interface such as web browser without worrying about the underlying cloud infrastructure such as network, operating systems, servers, storage or the platform on which the application is running. The user may only have to manage a limited user-specific application configuration settings. Thus, SaaS eliminates the need to install and run the application on each user’s own computer and simplifies the maintenance and support. Examples of SaaS are Microsoft Office 365 and Google Apps.

(2) **Cloud platform as a service (PaaS):** In this model the provider offers a computing platform such as programming language execution tools, web server, operating system, and database. Users can develop/run the softwares on the cloud without worrying about the cost or complexity of purchasing or maintaining them. PaaS automatically scales the underlying storage and computing resources to match the cloud user’s demands. Examples of PaaS are Heroku and Engine Yard.

(3) **Cloud infrastructure as a service (IaaS).** Cloud provider offers computers as physical/virtual machines and other resources including processing, raw and file storage, networks, computing resources, firewalls. The users get these resources from large pools installed in data centers. The users install operating systems and application software images on their computers but in this model users are responsible for their repair/maintenance and pay the bills to the provider on utility computing basis. Examples of IaaS are Amazon Elastic Compute Cloud and Rackspace Cloud.

4.4. Deployment Models

(1) **Private cloud** This cloud infrastructure is operated exclusively for an organization and may be managed by the organization or a third party. The private cloud project raises security questions which must be handled carefully.

(2) **Community cloud** This infrastructure is shared among several organizations having common interest such as mission, security requirements, policy, and agreement
considerations). It could be administrated internally or externally.

(3) **Public cloud** This infrastructure is made available to the general public or a large industry group and is owned by an organization selling cloud services. Usually the providers own and manage the infrastructure and grant access through Internet.

(4) **Hybrid cloud** The infrastructure is composed of two or more clouds (private, community, or public) that remain unique entities but are bound together. Through hybridization, the users can obtain local usability without Internet dependence.
5.1. System Overview

To build dependable cloud computing systems, we propose a reconfigurable distributed virtual machine (RDVM) infrastructure, which leverages the virtualization technologies to facilitate failure-aware cloud resource management. Anomaly Detector is a key component in this infrastructure. A RDVM, as illustrated in figure 5.1, consists of a set of virtual machines running on top of physical servers in a cloud. Each VM encapsulates execution states of cloud services and running client applications. It is the basic unit of management for RDVM construction and reconfiguration. Each cloud server hosts multiple virtual machines. These virtual machines multiplex resources of the underlying physical server. The virtual machine monitor (VMM, also called hypervisor) is a thin layer that manages hardware resources and exports a uniform interface to the upper guest [54].

This chapter is also presented in my accepted publications [41, 45]

---

Figure 5.1. A dependable cloud computing infrastructure.
When a client application is submitted with its computation and storage requirement to the cloud, the *cloud coordinator* evaluates qualifications of available cloud servers. It selects one or a set of them for the application, initiates the creation of VMs on them, and then dispatches the application instances for execution. Virtual machines on a cloud server are managed locally by a *RDVM daemon*, which is also responsible for communication with *resource manager*, *anomaly detector* and *cloud coordinator*. The RDVM daemon monitors the health status of the corresponding cloud server, collects runtime performance data of local VMs, and sends them to the *anomaly detector*, which characterizes cloud behaviors, identifies possible failure states, and reports the detected failures to cloud operators for verification. The verified detections will be input back to the anomaly detector for adaptation. Based on the performance data and failure reports, the *resource manager* analyzes the workload distribution, online availability, and allocated and available cloud resources, and then makes RDVM reconfiguration decisions. The anomaly detector and resource manager form a closed feedback control loop to deal with dynamics and uncertainty of the cloud computing environment.

To identify failures, hybrid anomaly detector needs the runtime cloud performance data. The performance data collected periodically by the RDVM daemons include the application execution status and the runtime utilization information of various virtualized resources on virtual machines. RDVM daemons also work with hypervisors to record the performance of hypervisors and monitor the utilization of underlying hardware resources/devices. These data and information from multiple system levels (i.e., hardware, hypervisor, virtual machine, RDVM, and the cloud) are valuable for accurate assessment of the cloud health and for detecting and pinpointing failures. They constitute the health-related cloud performance dataset, which is explored by Anomaly detector.

5.2. System Design

In this section, we present the design details of our system. We focus on the design of the anomaly detector. We first describe the performance metric extraction scheme followed by the adaptive failure detection scheme.
5.2.1. Cloud Metric Extraction

Runtime performance data are collected across a cloud computing system and the data transformation component assembles the data and compiles them in a uniform format. A metric (feature) in the runtime performance dataset refers to any individual measurable variable of a cloud server or network being monitored. It can be a statistic of usage of hardware, virtual machines, or cloud applications. In production cloud computing systems, usually hundreds of performance metrics are monitored and measured. The large metric dimension and the overwhelming volume of cloud performance data make the data model extremely complex. Moreover, the existence of interacting metrics and external environmental factors introduce measurement noises in the collected data.

To achieve efficient and accurate failure detection, the first step is to extract the most relevant performance metrics to characterize a cloud’s behavior and health. This step transforms the cloud performance data to a new metric space with only the most important attributes preserved. Given the input cloud performance dataset $D$ including $L$ records of $N$ metrics $M = \{m_i, i = 1, \ldots, N\}$, and the classification variable $c$, metric extraction is to find from the $N$-dimensional measurement space, $\mathbb{R}^N$, a subspace of $n$ metrics (subset $S$), $\mathbb{R}^n$, that optimally characterizes $c$. For a two-class failure detection, the value of variable $c$ can be either 0 or 1 representing the “normal” or “failure” state. In a multi-class failure detection, each failure type corresponds to a positive number that variable $c$ can take.

Anomaly detector first extracts those metrics, which jointly have the highest dependency on the class $c$. To achieve this goal, Anomaly detector quantifies the mutual dependence of a pair of metrics, say $m_i$ and $m_j$. Their mutual information (MI) [12] is defined as $I(m_i; m_j) = H(m_i) + H(m_j) - H(m_i m_j)$, where $H(\cdot)$ refers to the Shannon entropy [61]. Metrics of the cloud performance data usually take discrete values. The marginal probability $p(m_i)$ and the probability mass function $p(m_i, m_j)$ can be calculated using the collected dataset. Then, the MI of $m_i$ and $m_j$ is computed as $I(m_i; m_j) = \sum_{m_i \in M} \sum_{m_j \in M} p(m_i, m_j) \log \frac{p(m_i, m_j)}{p(m_i)p(m_j)}$. We choose the mutual information for metric extraction because of its capability of measuring any type of relationship between variables.
and its invariance under space transformation.

Anomaly detector applies two criteria to extract cloud metrics: finding the metrics that have high relevance with the class $c$ (maximal relevance criterion) and have low mutual redundancy between each other (minimal redundancy criterion). The metric relevance and redundancy are quantified as follows.

\begin{align}
\text{relevance} &= \frac{1}{|S|} \sum_{m_i \in S} I(m_i; c), \\
\text{redundancy} &= \frac{1}{|S|^2} \sum_{m_i, m_j \in S} I(m_i; m_j),
\end{align}

where $|S|$ is the cardinality of the extracted subset of cloud metrics $S$. The $N$ metrics in the metric set $M$ defines a $2^N$ search space. Finding the optimal metric subset is NP-hard [65]. To extract the near-optimal metrics satisfying Criteria (60), we apply the incremental metric search algorithm [18].

From our experiments, we find the resulting subset $S$ still contains many cloud metrics. Therefore, we extract the cloud metrics further by applying metric space separation. This is done by independent component analysis (ICA) method [31]. ICA is particularly suitable for separating a multivariate signal of the non-Gaussian source. Principle component analysis (PCA) [48] could be used for dimension reduction, but for this application, ICA works better than PCA.
CHAPTER 6

APPLICATIONS OF AAD AND HAD TO ANOMALY DETECTION IN CLOUD COMPUTING

Cloud computing has become increasingly popular by obviating the need for users to own and maintain complex computing infrastructure. However, due to their inherent complexity and large scale, production cloud computing systems are prone to various runtime problems caused by hardware and software failures. In this chapter we discuss the performance evaluations of our AAD and HAD algorithms as an application to detect anomalies and make the cloud system self dependable.

6.1. Experiment Settings

The cloud computing system consists of 362 servers, which are connected by gigabit Ethernet. The cloud servers are equipped with two to four Intel Xeon or AMD Opteron cores and 2.5 to 8 GB of RAM. We have installed Xen 3.1.2 hypervisors on the cloud servers. The operating system on a virtual machine is Linux 2.6.18 as distributed with Xen 3.1.2. Each cloud server hosts up to eight VMs. A VM is assigned up to two VCPUs, among which the number of active ones depends on applications. The amount of memory allocated to a VM is set to 512 MB. We run the RUBiS [7] distributed online service benchmark and MapReduce [14] jobs as cloud applications on VMs. The applications are submitted to the cloud computing system through a web based interface. We have also developed a anomaly injection program, which is able to randomly inject four major types with 17 sub-types of anomalies to cloud servers. They mimic anomalies from CPU, memory, disk, and network.

We exploit the third-party monitoring tools, such as SYSSTAT [64] to collect runtime performance data in Dom0 and a modified PERF [13] to obtain the values of performance counters from the Xen hypervisor on each cloud server. In total, 518 metrics are profiled 10 times per hour for one month (in Summer 2011). They cover the statistics of every

Parts of this chapter are presented in their entirety in my accepted publication [46], [41], [45], and [40] with Springer publication.
component of a cloud server, including CPU usage, process creation, task switching activity, memory and swap space utilization, paging and page anomaly, interrupts, network activity, I/O and data transfer, power management, and more. In total, about 601.4 GB health-related performance data were collected and recorded from the cloud in that period of time. Among all the metrics, 112 of them display zero variance, which provides no contribution to anomaly detection. After removing them, we have 406 non-constant metrics left. Then, AAD detector applies ICA to extract the cloud metrics further. The new metric space can present the original dataset in a more concise way. Figure 6.2 shows the results after performing the cloud metric extraction on the 14 metrics extracted in the preceding step. From the figure, we can see that the first three metrics can capture most (i.e., 81.3%) of the variance from the original cloud performance data. Thus, the dimension of the cloud metric space is further reduced to three.

6.2. Cloud Metric Extraction Results

List of major features of performance data collected by health monitoring tools in an institute-wide cloud computing system [25] are given in Table 6.1 - 6.3. The anomaly detector uses mutual information (MI) to quantify the relevance and redundancy of pair-wise cloud metrics. For $N$ cloud metrics, the algorithm needs to compute $\binom{N}{2}$ mutual information values. After removing the zero-variance metrics, we have $N = 406$. In total, $\binom{406}{2} = 82,215$
<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU Statistics</strong></td>
<td></td>
</tr>
<tr>
<td>PROC/S</td>
<td>Total number of tasks created per second</td>
</tr>
<tr>
<td>Cswch/s</td>
<td>Total number of context switches per second</td>
</tr>
<tr>
<td>%user</td>
<td>Percentage of CPU utilization that occurred while executing at the user level (application). Note that this field includes time spent running virtual processors</td>
</tr>
<tr>
<td>%nice</td>
<td>CPU utilization that occurred while executing at user level with nice priority</td>
</tr>
<tr>
<td>%system</td>
<td>Percentage of CPU utilization that occurred while executing at the system level (kernel). Note that this field includes time spent servicing interrupts and soft IRQs</td>
</tr>
<tr>
<td>%iowait</td>
<td>Percentage of time that CPU(s) were idle during which the system had an outstanding disk I/O request</td>
</tr>
<tr>
<td>%idle</td>
<td>Percentage of time that CPU(s) were idle and the system did not have an outstanding disk I/O request</td>
</tr>
<tr>
<td>Intr/s</td>
<td>Shows the total number of interrupts received per second by CPU</td>
</tr>
<tr>
<td><strong>SWAP Statistics</strong></td>
<td></td>
</tr>
<tr>
<td>PSWPIN/s</td>
<td>Total number of swap pages the system brought in per second</td>
</tr>
<tr>
<td>PSWPOUT/s</td>
<td>Total number of swap pages the system brought out per second</td>
</tr>
<tr>
<td>PGPGIN/s</td>
<td>Total number of kilobytes the system paged in from disk per second</td>
</tr>
<tr>
<td>PGPOUT/s</td>
<td>Total number of kilobytes the system paged out to disk per second</td>
</tr>
<tr>
<td>anomaly/s</td>
<td>Number of page anomalies (major+minor) made by the system per second</td>
</tr>
<tr>
<td>Majfit/s</td>
<td>Number of major anomalies the system made per second, those which have required loading a memory page from disk</td>
</tr>
<tr>
<td><strong>I/O Requests</strong></td>
<td></td>
</tr>
<tr>
<td>Tps</td>
<td>Total number of transfers per sec that were issued to physical devices. A transfer is an I/O request to a physical device. Multiple logical requests can be combined into a single I/O request to device. A transfer is of indeterminate size.</td>
</tr>
</tbody>
</table>
Table 6.2. I/O requests (continued), memory page statistics, %memused

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rtps</td>
<td>Total number of read quests issued to physical devices</td>
</tr>
<tr>
<td>Wtps</td>
<td>Total number of write requests per second issued to physical devices</td>
</tr>
<tr>
<td>Bread/s</td>
<td>Total amount of data read from the devices in blocks per second. Blocks are equivalent to sectors with 2.4 kernels and newer and therefore have a size of 512 bytes. Older kernels, a block is of indeterminate size</td>
</tr>
<tr>
<td>Bwrtn/s</td>
<td>Total amount of data written to devices in blocks per second</td>
</tr>
<tr>
<td>Rd_sec/s</td>
<td>Number of sectors read from the device. The size of a sector is 512 bytes</td>
</tr>
<tr>
<td>Wr_sec/s</td>
<td>Number of sectors written to the device. The size of a sector is 512 bytes</td>
</tr>
<tr>
<td>Frmpg/s</td>
<td>Number of memory pages freed by the system per second. A negative value represents a number of pages allocated by the system. Note that a page has a size of 4 KB or 8 KB according to the machine architecture</td>
</tr>
<tr>
<td>Bufpg/s</td>
<td>Number of additional memory pages used as buffers by the system per sec. A negative value means fewer pages used as buffer by system</td>
</tr>
<tr>
<td>Campg/s</td>
<td>Number of additional memory pages cached by the system per second. A negative value means fewer pages used as buffer by the system</td>
</tr>
<tr>
<td>KBMEMFREE</td>
<td>Amount of free memory available in kilobytes</td>
</tr>
<tr>
<td>KBMEMUSED</td>
<td>Amount of used memory in kb excluding memory used by kernel</td>
</tr>
<tr>
<td>%memused</td>
<td>Percentage of used memory</td>
</tr>
<tr>
<td>KBBUFFERS</td>
<td>Amount of memory used as buffers by the kernel in kilobytes</td>
</tr>
<tr>
<td>KBCACHED</td>
<td>Amount of memory used to cache data by the kernel in kilobytes</td>
</tr>
<tr>
<td>KBSWPFREE</td>
<td>Amount of free swap space in kilobytes</td>
</tr>
<tr>
<td>KBSWPUSED</td>
<td>Amount of used swap space in kilobytes</td>
</tr>
<tr>
<td>KBSWPCAD</td>
<td>Amount of cached swap memory in kilobytes. This is memory that once was swapped out, is swapped back in but still also is in the swap area (if memory is needed it doesn’t need to be swapped out again because it is already in the swap area This saves I/O)</td>
</tr>
<tr>
<td><strong>Network Statistics</strong></td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>%SWPUSED</td>
<td>Percentage of used swap space</td>
</tr>
<tr>
<td>RXPCK/S</td>
<td>Total number of packets received per second</td>
</tr>
<tr>
<td>RXBYT/S</td>
<td>Total number of bytes received per second</td>
</tr>
<tr>
<td>TXPCK/S</td>
<td>Total number of packets transmitted per second</td>
</tr>
<tr>
<td>TXBYT/S</td>
<td>Total number of bytes transmitted per second</td>
</tr>
<tr>
<td>RXCMP/S</td>
<td>Number of compressed packets received per second</td>
</tr>
<tr>
<td>RXERR/S</td>
<td>Total number of bad packets received per second</td>
</tr>
<tr>
<td>TXERR/S</td>
<td>Total number of errors that happened per second while transmitting packets</td>
</tr>
<tr>
<td>COLL/S</td>
<td>Number of collisions that happened per second while transmitting packets</td>
</tr>
<tr>
<td>RXDROP/S</td>
<td>Number of received packets dropped per second because of lack of space in Linux buffers</td>
</tr>
<tr>
<td>TXCARR/S</td>
<td>Number of carrier-errors that happened per sec while transmitting packets</td>
</tr>
<tr>
<td>RXFRAM/S</td>
<td>Number of frame alignment errors that happened per sec on received packets</td>
</tr>
<tr>
<td>RXFIFO/S</td>
<td>Number of FIFO overrun errors that happened per sec on received packets</td>
</tr>
<tr>
<td>TXFIFO/S</td>
<td>Number of FIFO overrun errors happened per sec on transmitted packets</td>
</tr>
<tr>
<td>TOTSCK</td>
<td>Total number of sockets used by the system</td>
</tr>
<tr>
<td>TCPSCK</td>
<td>Number of TCP sockets currently in use</td>
</tr>
<tr>
<td>UDPSCK</td>
<td>Number of UDP sockets currently in use</td>
</tr>
<tr>
<td>RAWSCK</td>
<td>Number of RAW sockets currently in use</td>
</tr>
<tr>
<td>IP-FRAG</td>
<td>Number of IP fragments currently in use</td>
</tr>
<tr>
<td>RXMCST/S</td>
<td>Number of multicast packets received per second</td>
</tr>
</tbody>
</table>

MI values are computed. A smaller MI values infers the corresponding pair of metrics share less information. Then our anomaly detector calculates the redundancy and relevance among the cloud metrics, and thus their dependency according to Equation (60). For ease of comparison and visualization, we calculate the inverse of the dependency, i.e. \( \text{redundancy} - \)
Table 6.4. 14 most prominent metrics

<table>
<thead>
<tr>
<th>cswch/s</th>
<th>pswpout/s</th>
<th>pgscank/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>%vmeff</td>
<td>%system</td>
<td>intr/s</td>
</tr>
<tr>
<td>%iowait</td>
<td>ITLB-loads</td>
<td>kbbmemused</td>
</tr>
<tr>
<td>kbbuffers</td>
<td>wr_sec/s</td>
<td>dev253-1</td>
</tr>
<tr>
<td>rxpck/s</td>
<td>totsck</td>
<td></td>
</tr>
</tbody>
</table>

(relevance), and search for the metrics with minimal values. Figure 6.1 shows the normalized (redundancy - relevance) values of the 406 non-constant metrics. By setting a threshold $\lambda = 0.15$, anomaly detector extracts those cloud metrics whose normalized (redundancy - relevance) $\leq \lambda$. In total, 14 metrics satisfy this condition. They are shown in the Table 6.4.

6.2.1. AAD Performance Evaluation

We have implemented a proof-of-concept prototype of our AAD system and tested it in a cloud computing environment on campus. In this section, we present the experimental results. We evaluate the performance of anomaly detection function by measuring detection sensitivity and specificity.

\[
\text{detection sensitivity} = \frac{\text{number of detected anomalies}}{\text{total number of anomalies}}
\]

\[
\text{detection specificity} = \frac{\text{number of detected normal states}}{\text{total number of normal states}}
\]

The AAD detector identifies possible anomalies in the cloud performance data. It adapts itself by learning the verified detection results and observed but undetected failure events reported from the cloud operators. Figures 6.3 and 6.4 show AAD’s adaptation of the data description contours for anomaly detection after 78 and 109 verified detection results are learned by the AAD detector. Only 1% of two ICA components normal data points are plotted in the figure for better readability. From the figures, we can see as more verified detection results are exploited, the contours become tighter, which improves the accuracy of anomaly detection. Data distribution of anomaly detection results with self-evolvement III
Figure 6.3. Contour for anomaly detection after AAD-adaptation I.

Figure 6.4. Contour for anomaly detection after AAD-adaptation III.

are shown in figure 6.5 in three dimensions. Figure 6.6 depicts the detection performance after three selected rounds of adaptation. 78, 66, and 43 verified detection results are learned by the anomaly detector during the AAD-adaptation I, II, and III, respectively. The figure shows both the detection sensitivity and the detection specificity improve as the AAD detector adapts. After AAD-adaptation III is applied, the AAD detector achieves 92.1% detection sensitivity and 83.8% detection specificity. These results indicate that AAD is well suitable for building self dependable anomaly detection computing systems. We have also compared AAD with anomaly detectors using some advanced learning algorithms. Subspace Regularization [72] performs better than other smoothness-based methods, including Gaussian random Field [74] and manifold regularization [3]. In our experiments, the anomaly detector using subspace regularization achieves 67.8% sensitivity. The ensemble of Bayesian sub-models and decision tree classifiers that we proposed in [25] could only have 72.5% detection sensitivity. Our AAD algorithm can achieve 92.1% detection sensitivity and 83.8% detection specificity, which are much higher than the other detectors. To make an anomaly detection, it takes 7.26 seconds on average for a control node in the cloud to extract cloud performance metrics, create the hypersphere, and make anomaly detections. The following adaptation steps are even more lightweight, taking an average of 2.17 seconds to update the hypersphere and identify anomalies.
Figure 6.5. Anomaly detection results with self-evolvement III. Only 10% of the detected normal data points are plotted due to the size of the figure and the readability.

Figure 6.6. Sensitivity and specificity of anomaly detection improves with AAD’s self adaptations.
6.2.2. Conclusion

We presented AAD, an adaptive anomaly detection framework with mechanisms. Different from other anomaly detection approaches, AAD does not require a prior failure history and it can self-adapt by learning from observed failure events at runtime. Thereby, it is capable of finding anomalies not yet seen in the past. Based on the experiments on cloud performance data, AAD detects possible anomalies, which are verified by the cloud operators. They are confirmed as either true anomaly or normal states. It adapts itself by recursively learning from these newly verified detection results to refine anomaly detections. Meanwhile, it exploits the observed but undetected anomaly records reported by the system administrators to identify new types of anomalies. The proposed AAD framework with mechanisms in this research can also aid anomaly prediction. Complementing existing anomaly prediction methods, results from this algorithm can be utilized to determine the potential localization of anomalies by analyzing the runtime streaming performance data.

6.2.3. HAD Performance Evaluation

We apply our hybrid self-evolving anomaly detection framework using one-class and two-class support vector machines for anomaly detection. System design, experimental settings and cloud metric extraction for HAD are exactly same as AAD algorithm. Experimental results in an institute wide cloud computing system show that the detection accuracy of the algorithm improves as it evolves and it can achieve 92.1% detection sensitivity and 83.8% detection specificity, which makes it well suitable for building highly autonomous systems. We evaluate the performance of HAD framework by detection sensitivity and specificity. Our anomaly detector identifies possible anomalies in the cloud performance data. It adapts itself by learning the verified detection results and observed but undetected anomaly events reported from the cloud operators. Figures 6.7 and 6.8 show anomaly detector’s adaptation of the data description contours for anomaly detection (green ‘*’ and red ‘+’ stand for normal and anomaly class data respectively). Less than 1% of the training data points are plotted in the figure for better readability. From the figures, we can see as more verified detection results are exploited, the contours become tighter, which improves the accuracy of anomaly
Figure 6.7. Contours of decision function during Adaption-I

Figure 6.8. Contours of decision function during Adaption-II

Figure 6.9. Credibility scores improvement and stabilization of 1-class and 2-class SVM during hybridization

Figure 6.10. Sensitivity and specificity of hybrid anomaly detection algorithm (HAD) with adaptation

detection. Figure 6.9 shows that credibility of one and two class SVM gets stabilize over period of time. Figure 6.10 depicts the detection performance after three selected rounds of adaptation by hybrid anomaly detector I, II, and III, respectively. The figure shows both the detection sensitivity and the detection specificity improve as the detector adapts. After adaptation III is applied, the anomaly detector achieves 92.1% detection sensitivity and 83.8% detection specificity. These results indicate that our anomaly detector is well suitable for building dependable cloud computing systems.
We compared our algorithm with anomaly detectors with modern advanced learning algorithms. Subspace regularization [72] performs better than other smoothness-based methods, including Gaussian random field [74] and manifold regularization [3]. In our experiments, the anomaly detector using subspace regularization achieves 67.8% sensitivity. The ensemble of Bayesian sub-models and decision tree classifiers that we proposed in [25] could only have 72.5% detection sensitivity. Our anomaly detector can achieve 92.1% detection sensitivity and 83.8% detection specificity, which are much higher than the other detectors.

6.2.4. Conclusion

These days large-scale and complex online anomaly detection mechanisms are susceptible to software and hardware failures and human mistakes, which significantly affect the system dependability and performance. In this research, we employ hybrid anomaly detection mechanism based on 1-class and 2-class SVM for adaptive anomaly detection. Different from other anomaly detection approaches, it does not require a prior anomaly history and it can self-adapt by learning from observed anomaly events at runtime. Thereby, it is capable of finding failures not yet seen in the past. Based on the collected performance data, it detects possible anomalies, which are verified by the data analysts. We note that even with the most advanced learning methods, the accuracy of anomaly detection cannot reach 100%. Reactive approaches, such as checkpointing and redundant execution, should be included to handle mis-detections. We plan to integrate the proactive and reactive anomaly management approaches to achieve even higher system dependability.
CHAPTER 7

CONCLUSION

Semi-supervised learning (SSL) is the most practical and useful technique in machine learning. It stands in between supervised and unsupervised learning. It exploits most of the unlabeled and few labeled data and is similar to humans way of learning according to cognitive science perspective. We proposed two new self evolving semi-supervised anomaly detection algorithms as applications for building dependable and autonomic systems. adaptive anomaly detection (AAD) algorithm employs data description based on hypersphere whose position and shape keeps improving in order to enclose the normal positive class data during retraining. Hybrid anomaly detection (HAD) algorithm is based on one-class and two-class support vector machines. To speedup the retraining phase, it employs data set selection to preserve and update only the most crucial support vectors required to determine the classification function. AAD and HAD algorithms are different from other anomaly detection approaches which do not require prior failure history and can self adapt by learning from observed anomalies at runtime. Thereby, they are capable of finding failures not yet seen in the past. Based on the incoming performance data, AAD and HAD detects possible failures, which are verified by the data analysts. They are confirmed as either true failures or normal states. Our algorithms adapt themselves by recursively learning from these newly verified detection results to refine future detections. Meanwhile, AAD and HAD exploit the observed but undetected anomalous records reported by the data analysts to identify new types of anomalies. We plan to explore other advanced statistical learning methods for anomaly detection and automated adaptation for dependability assurance in secure computing environments. We note that even with the most advanced learning methods, the accuracy of failure detection cannot reach 100%. Reactive approaches, such as check pointing and redundant execution, should be included to handle mis-detections. We plan to integrate the proactive and reactive failure management approaches to achieve even higher system dependability.
BIBLIOGRAPHY


[40] J. Liu, S. Fu, and H. Pannu, A hybrid anomaly detection framework in cloud computing using one-class and two-class support vector machines, Advanced Data Mining and Applications (2012), Springer Publication.


[45] H.S. Pannu, S. Fu, and J. Liu, AFD: Adaptive failure detection system for cloud comput-
ing infrastructures, IEEE. International Performance Computing and Communications Conference (2012).


